Contents

Preface

Audience v
Documentation Accessibility v
Related Documents vi
Conventions vi

Changes in This Release for Oracle Streams Extended Examples

Changes in Oracle Database 12c Release 1 (12.1) vii

1 Simple Single-Source Replication Example

1.1 Overview of the Simple Single-Source Replication Example 1-1
1.2 Prerequisites 1-2
1.3 Create Queues and Database Links 1-3
1.4 Configure Capture, Propagation, and Apply for Changes to One Table 1-5
1.5 Make Changes to the hr.jobs Table and View Results 1-9

2 Single-Source Heterogeneous Replication Example

2.1 Overview of the Single-Source Heterogeneous Replication Example 2-1
2.2 Prerequisites 2-3
2.3 Create Queues and Database Links 2-5
2.4 Example Scripts for Sharing Data from One Database 2-10
   2.4.1 Simple Configuration for Sharing Data from a Single Database 2-10
   2.4.2 Flexible Configuration for Sharing Data from a Single Database 2-23
2.5 Make DML and DDL Changes to Tables in the hr Schema 2-37
2.6 Add Objects to an Existing Oracle Streams Replication Environment 2-38
2.7 Make a DML Change to the hr.employees Table 2-47
2.8 Add a Database to an Existing Oracle Streams Replication Environment 2-48
2.9 Make a DML Change to the hr.departments Table 2-57
## 3 N-Way Replication Example

- **3.1 Overview of the N-Way Replication Example** 3-1
- **3.2 Prerequisites** 3-3
- **3.3 Create the hrmutl Schema at the mult1.example.com Database** 3-4
- **3.4 Create Queues and Database Links** 3-5
- **3.5 Example Script for Configuring N-Way Replication** 3-12
- **3.6 Make DML and DDL Changes to Tables in the hrmutl Schema** 3-34

## 4 Single-Database Capture and Apply Example

- **4.1 Overview of the Single-Database Capture and Apply Example** 4-1
- **4.2 Prerequisites** 4-2
- **4.3 Set Up the Environment** 4-3
- **4.4 Configure Capture and Apply** 4-5
- **4.5 Make DML Changes, Query for Results, and Dequeue Messages** 4-12

## 5 Logical Change Records with LOBs Example

- **5.1 Example Script for Constructing and Enqueuing LCRs Containing LOBs** 5-1

## 6 Rule-Based Application Example

- **6.1 Overview of the Rule-Based Application** 6-1
- **6.2 Using Rules on Nontable Data Stored in Explicit Variables** 6-2
- **6.3 Using Rules on Data in Explicit Variables with Iterative Results** 6-7
- **6.4 Using Partial Evaluation of Rules on Data in Explicit Variables** 6-10
- **6.5 Using Rules on Data Stored in a Table** 6-16
- **6.6 Using Rules on Both Explicit Variables and Table Data** 6-21
- **6.7 Using Rules on Implicit Variables and Table Data** 6-27
- **6.8 Using Event Contexts and Implicit Variables with Rules** 6-34
- **6.9 Dispatching Problems and Checking Results for the Table Examples** 6-41

## Index
Preface

*Oracle Streams Extended Examples* includes detailed examples that use Oracle Streams features.

This Preface contains these topics:

- **Audience**
- **Documentation Accessibility**
- **Related Documents**
- **Conventions**

**Audience**

*Oracle Streams Extended Examples* is intended for database administrators who create and maintain Oracle Streams environments. These administrators perform one or more of the following tasks:

- Plan for an Oracle Streams environment
- Configure an Oracle Streams environment
- Administer an Oracle Streams environment
- Monitor an Oracle Streams environment

To use this document, you must be familiar with relational database concepts, SQL, distributed database administration, general Oracle Streams concepts, Advanced Queuing concepts, PL/SQL, and the operating systems under which you run an Oracle Streams environment.

**Documentation Accessibility**


**Access to Oracle Support**

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

For more information, see these Oracle resources:

- Oracle Streams Concepts and Administration
- Oracle Streams Replication Administrator's Guide
- Oracle Database Concepts
- Oracle Database Administrator's Guide
- Oracle Database SQL Language Reference
- Oracle Database PL/SQL Packages and Types Reference
- Oracle Database PL/SQL Language Reference
- Oracle Database Utilities
- Oracle Database Heterogeneous Connectivity User's Guide
- Oracle Streams online Help for the Oracle Streams tool in Oracle Enterprise Manager Cloud Control

Many of the examples in this book use the sample schemas. See Oracle Database Sample Schemas for information about these schemas.

Conventions

The following text conventions are used in this document:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>boldface</strong></td>
<td>Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.</td>
</tr>
<tr>
<td><em>italic</em></td>
<td>Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.</td>
</tr>
<tr>
<td>monospace</td>
<td>Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.</td>
</tr>
</tbody>
</table>
Changes in This Release for Oracle Streams Extended Examples

This preface contains:

• Changes in Oracle Database 12c Release 1 (12.1)

Changes in Oracle Database 12c Release 1 (12.1)

The following are changes in Oracle Streams Extended Examples for Oracle Database 12c Release 1 (12.1).

Deprecated Features

• Oracle Streams is deprecated in Oracle Database 12c Release 1 (12.1). Use Oracle GoldenGate to replace all replication features of Oracle Streams.

Oracle Streams does not support any Oracle Database features added in Oracle Database 12c Release 1 (12.1) or later releases.

Note:

Oracle Database Advanced Queuing is independent of Oracle Streams and continues to be enhanced.

See Also:

The Oracle GoldenGate documentation
Simple Single-Source Replication Example

This chapter illustrates an example of a simple single-source replication environment that can be constructed using Oracle Streams.

This chapter contains these topics:

- Overview of the Simple Single-Source Replication Example
- Prerequisites
- Create Queues and Database Links
- Configure Capture, Propagation, and Apply for Changes to One Table
- Make Changes to the hr.jobs Table and View Results

1.1 Overview of the Simple Single-Source Replication Example

The example in this chapter illustrates using Oracle Streams to replicate data in one table between two databases. A capture process captures data manipulation language (DML) and data definition language (DDL) changes made to the jobs table in the hr schema at the str1.example.com Oracle database, and a propagation propagates these changes to the str2.example.com Oracle database. Next, an apply process applies these changes at the str2.example.com database. This example assumes that the hr.jobs table is read-only at the str2.example.com database.

Figure 1-1 provides an overview of the environment.
1.2 Prerequisites

The following prerequisites must be completed before you begin the example in this chapter.

- Set the following initialization parameters to the values indicated:
  - `GLOBAL_NAMES`: This parameter must be set to `TRUE` at each database that is participating in your Oracle Streams environment.
  - `COMPATIBLE`: This parameter must be set to `10.2.0` or higher at each database that is participating in your Oracle Streams environment.
  - `STREAMS_POOL_SIZE`: Optionally set this parameter to an appropriate value for each database in the environment. This parameter specifies the size of the Oracle Streams pool. The Oracle Streams pool stores messages in a buffered queue and is used for internal communications during parallel capture and apply. When the `MEMORY_TARGET`, `MEMORY_MAX_TARGET`, or `SGA_TARGET` initialization parameter is set to a nonzero value, the Oracle Streams pool size is managed automatically.

  
  **See Also:**
  
  *Oracle Streams Replication Administrator's Guide* for information about other initialization parameters that are important in an Oracle Streams environment

- Any database producing changes that will be captured must be running in `ARCHIVELOG` mode. In this example, changes are produced at `str1.example.com`, and so `str1.example.com` must be running in `ARCHIVELOG` mode.

  
  **See Also:**
  
  *Oracle Database Administrator's Guide* for information about running a database in `ARCHIVELOG` mode

- Configure your network and Oracle Net so that the `str1.example.com` database can communicate with the `str2.example.com` database.

  
  **See Also:**
  
  *Oracle Database Net Services Administrator's Guide*

- Create an Oracle Streams administrator at each database in the replication environment. In this example, the databases are `str1.example.com` and `str2.example.com`. This example assumes that the user name of the Oracle Streams administrator is `strmadmin`. 
1.3 Create Queues and Database Links

Complete the following steps to create queues and database links for an Oracle Streams replication environment that includes two Oracle databases.

1. Show Output and Spool Results
2. Create the ANYDATA Queue at str1.example.com
3. Create the Database Link at str1.example.com
4. Set Up the ANYDATA Queue at str2.example.com
5. Check the Spool Results

Note:

If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line after this note to the next "END OF SCRIPT" line into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to all of the databases in the environment.

/**************************** BEGINNING OF SCRIPT ****************************

Show Output and Spool Results
Run SET ECHO ON and specify the spool file for the script. Check the spool file for errors after you run this script.

*/

SET ECHO ON
SPOOL streams_setup_simple.out

/*

Create the ANYDATA Queue at str1.example.com
Connect as the Oracle Streams administrator at the database where you want to capture changes. In this example, that database is str1.example.com.

*/

CONNECT strmadmin@str1.example.com

/*
Run the `SET_UP_QUEUE` procedure to create a queue named `streams_queue` at `str1.example.com`. This queue will function as the ANYDATA queue by holding the captured changes that will be propagated to other databases. Running the `SET_UP_QUEUE` procedure performs the following actions:

- Creates a queue table named `streams_queue_table`. This queue table is owned by the Oracle Streams administrator (`strmadmin`) and uses the default storage of this user.
- Creates a queue named `streams_queue` owned by the Oracle Streams administrator (`strmadmin`).
- Starts the queue.

```
EXEC DBMS_STREAMS_ADM.SET_UP_QUEUE();
```

Create the Database Link at `str1.example.com`
Create the database link from the database where changes are captured to the database where changes are propagated. In this example, the database where changes are captured is `str1.example.com`, and these changes are propagated to `str2.example.com`.

```
ACCEPT password PROMPT 'Enter password for user: ' HIDE
CREATE DATABASE LINK str2.example.com CONNECT TO strmadmin IDENTIFIED BY &password USING 'str2.example.com';
```

Set Up the ANYDATA Queue at `str2.example.com`
Connect as the Oracle Streams administrator at `str2.example.com`.

```
CONNECT strmadmin@str2.example.com
```

Run the `SET_UP_QUEUE` procedure to create a queue named `streams_queue` at `str2.example.com`. This queue will function as the ANYDATA queue by holding the changes that will be applied at this database. Running the `SET_UP_QUEUE` procedure performs the following actions:

- Creates a queue table named `streams_queue_table`. This queue table is owned by the Oracle Streams administrator (`strmadmin`) and uses the default storage of this user.
- Creates a queue named `streams_queue` owned by the Oracle Streams administrator (`strmadmin`).
- Starts the queue.
EXEC DBMS_STREAMS_ADM.SET_UP_QUEUE();

/*

Check the Spool Results
Check the streams_setup_simple.out spool file to ensure that all actions finished successfully after this script is completed.
*/

SET ECHO OFF
SPOOL OFF

/**************************** END OF SCRIPT ******************************/

1.4 Configure Capture, Propagation, and Apply for Changes to One Table

Complete the following steps to specify the capture, propagation, and apply definitions for the hr.jobs table using the DBMS_STEAMS_ADM package.

1. Show Output and Spool Results
2. Configure Propagation at str1.example.com
3. Configure the Capture Process at str1.example.com
4. Set the Instantiation SCN for the hr.jobs Table at str2.example.com
5. Configure the Apply Process at str2.example.com
6. Start the Apply Process at str2.example.com
7. Start the Capture Process at str1.example.com
8. Check the Spool Results

Note:
If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line after this note to the next "END OF SCRIPT" line into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to all of the databases in the environment.

/**************************** BEGINNING OF SCRIPT ******************************/

Show Output and Spool Results
Run SET ECHO ON and specify the spool file for the script. Check the spool file for errors after you run this script.

*/
/*

Configure Propagation at str1.example.com
Connect to str1.example.com as the strmadmin user.

*/

CONNECT strmadmin@str1.example.com

/*

Configure and schedule propagation of DML and DDL changes to the hr.jobs table from the queue at str1.example.com to the queue at str2.example.com.

*/

BEGIN
    DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES(
        table_name => 'hr.jobs',
        streams_name => 'strl_to_str2',
        source_queue_name => 'strmadmin.streams_queue',
        destination_queue_name => 'strmadmin.streams_queue@str2.example.com',
        include_dml => TRUE,
        include_ddl => TRUE,
        source_database => 'str1.example.com',
        inclusion_rule => TRUE,
        queue_to_queue => TRUE);
END;
/

/*

Configure the Capture Process at str1.example.com
Configure the capture process to capture changes to the hr.jobs table at str1.example.com. This step specifies that changes to this table are captured by the capture process and enqueued into the specified queue. This step also prepares the hr.jobs table for instantiation and enables supplemental logging for any primary key, unique key, bitmap index, and foreign key columns in this table. Supplemental logging places additional information in the redo log for changes made to tables. The apply process needs this extra information to perform certain operations, such as unique row identification and conflict resolution. Because str1.example.com is the only database where changes are captured in this environment, it is the only database where supplemental logging must be enabled for the hr.jobs table.

See Also:
Oracle Streams Replication Administrator's Guide

*/

BEGIN
DBMS_STREAMS_ADM.ADD_TABLE_RULES(
  table_name     => 'hr.jobs',
  streams_type   => 'capture',
  streams_name   => 'capture_simp',
  queue_name     => 'strmadmin.streams_queue',
  include_dml    => TRUE,
  include_ddl    => TRUE,
  inclusion_rule => TRUE);
END;
/
/*
Set the Instantiation SCN for the hr.jobs Table at str2.example.com
This example assumes that the hr.jobs table exists at both the str1.example.com database and the str2.example.com database, and that this table is synchronized at these databases. Because the hr.jobs table already exists at str2.example.com, this example uses the GET_SYSTEM_CHANGE_NUMBER function in the DBMS_FLASHBACK package at str1.example.com to obtain the current SCN for the source database. This SCN is used at str2.example.com to run the SET_TABLE_INSTANTIATION_SCN procedure in the DBMS_APPLY_ADM package. Running this procedure sets the instantiation SCN for the hr.jobs table at str2.example.com.
The SET_TABLE_INSTANTIATION_SCN procedure controls which LCRs for a table are ignored by an apply process and which LCRs for a table are applied by an apply process. If the commit SCN of an LCR for a table from a source database is less than or equal to the instantiation SCN for that table at a destination database, then the apply process at the destination database discards the LCR. Otherwise, the apply process applies the LCR.
In this example, both of the apply process at str2.example.com will apply transactions to the hr.jobs table with SCNs that were committed after SCN obtained in this step.

Note:
This example assumes that the contents of the hr.jobs table at str1.example.com and str2.example.com are consistent when you complete this step. Ensure that there is no activity on this table while the instantiation SCN is being set. You might want to lock the table at each database while you complete this step to ensure consistency. If the table does not exist at the destination database, then you can use export/import for instantiation.

$/
DECLARE
  iscn  NUMBER;         -- Variable to hold instantiation SCN value
BEGIN
  iscn := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER();
  DBMS_APPLY_ADM.SET_TABLE_INSTANTIATION_SCN@STR2.EXAMPLE.COM(
    source_object_name    => 'hr.jobs',
    source_database_name  => 'str1.example.com',
    instantiation_scn     => iscn);
END;
/
/*
Configure the Apply Process at str2.example.com

Connect to str2.example.com as the strmadmin user.

/*

CONNECT strmadmin@str2.example.com

*/

Configure str2.example.com to apply changes to the hr.jobs table.

/*

BEGIN

DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name => 'hr.jobs',
    streams_type => 'apply',
    streams_name => 'apply_simp',
    queue_name => 'strmadmin.streams_queue',
    include_dml => TRUE,
    include_ddl => TRUE,
    source_database => 'str1.example.com',
    inclusion_rule => TRUE);

END;
/

*/

Start the Apply Process at str2.example.com

Set the disable_on_error parameter to \n so that the apply process will not be disabled if it encounters an error, and start the apply process at str2.example.com.

/*

BEGIN

DBMS_APPLY_ADM.SET_PARAMETER(
    apply_name => 'apply_simp',
    parameter => 'disable_on_error',
    value => 'N'});

END;
/

BEGIN

DBMS_APPLY_ADM.START_APPLY(
    apply_name => 'apply_simp');

END;
/

Start the Capture Process at str1.example.com

Connect to str1.example.com as the strmadmin user.

/*

CONNECT strmadmin@str1.example.com

*/

Start the capture process at str1.example.com.
BEGIN
  DBMS_CAPTURE_ADM.START_CAPTURE(
    capture_name  => 'capture_simp');
END;
/

/*
Check the Spool Results
Check the streams_share_jobs.out spool file to ensure that all actions finished
successfully after this script is completed.
*/

SET ECHO OFF
SPOOL OFF

/*************************** END OF SCRIPT ******************************/

1.5 Make Changes to the hr.jobs Table and View Results

Complete the following steps to make DML and DDL changes to the hr.jobs table at str1.example.com and then confirm that the changes were captured at str1.example.com, propagated from str1.example.com to str2.example.com, and applied to the hr.jobs table at str2.example.com.

**Make Changes to hr.jobs at str1.example.com**
Make the following changes to the hr.jobs table.

CONNECT hr@str1.example.com
Enter password: password

UPDATE hr.jobs SET max_salary=9545 WHERE job_id='PR_REP';
COMMIT;

ALTER TABLE hr.jobs ADD(duties VARCHAR2(4000));

**Query and Describe the hr.jobs Table at str2.example.com**
After some time passes to allow for capture, propagation, and apply of the changes performed in the previous step, run the following query to confirm that the UPDATE change was propagated and applied at str2.example.com:

CONNECT hr@str2.example.com
Enter password: password

SELECT * FROM hr.jobs WHERE job_id='PR_REP';

The value in the max_salary Column should be 9545.

Next, describe the hr.jobs table to confirm that the ALTER TABLE change was propagated and applied at str2.example.com:

DESC hr.jobs

The duties column should be the last column.
Single-Source Heterogeneous Replication Example

This chapter illustrates an example of a single-source heterogeneous replication environment that can be constructed using Oracle Streams, as well as the tasks required to add new objects and databases to such an environment.

This chapter contains these topics:

• Overview of the Single-Source Heterogeneous Replication Example
• Prerequisites
• Create Queues and Database Links
• Example Scripts for Sharing Data from One Database
• Make DML and DDL Changes to Tables in the hr Schema
• Add Objects to an Existing Oracle Streams Replication Environment
• Make a DML Change to the hr.employees Table
• Add a Database to an Existing Oracle Streams Replication Environment
• Make a DML Change to the hr.departments Table

2.1 Overview of the Single-Source Heterogeneous Replication Example

This example illustrates using Oracle Streams to replicate data between four databases. The environment is heterogeneous because three of the databases are Oracle databases and one is a Sybase database. DML and DDL changes made to tables in the hr schema at the dbs1.example.com Oracle database are captured and propagated to the other two Oracle databases. Only DML changes are captured and propagated to the dbs4.example.com database, because an apply process cannot apply DDL changes to a non-Oracle database. Changes to the hr schema occur only at dbs1.example.com. The hr schema is read-only at the other databases in the environment.

Figure 2-1 provides an overview of the environment.
As illustrated in Figure 2-1, dbs1.example.com contains the following tables in the hr schema:

- countries
- departments
- employees
- job_history
- jobs
- locations
- regions

This example uses directed networks, which means that captured changes at a source database are propagated to another database through one or more intermediate databases. Here, the dbs1.example.com database propagates changes to the dbs2.example.com database through the intermediate database dbs2.example.com. This
configuration is an example of queue forwarding in a directed network. Also, the 
dbs1.example.com database propagates changes to the dbs2.example.com database, 
which applies the changes directly to the dbs4.example.com database through an Oracle 
Database Gateway.

Some of the databases in the environment do not have certain tables. If the database 
is not an intermediate database for a table and the database does not contain the 
table, then changes to the table do not need to be propagated to that database. For 
example, the departments, employees, job_history, and jobs tables do not exist at 
dbs3.example.com. Therefore, dbs2.example.com does not propagate changes to these 
tables to dbs3.example.com.

In this example, Oracle Streams is used to perform the following series of actions:

1. The capture process captures DML and DDL changes for all of the tables in the hr 
schema and enqueues them at the dbs1.example.com database. In this example, 
changes to only four of the seven tables are propagated to destination databases, 
but in the example that illustrates "Add Objects to an Existing Oracle Streams 
Replication Environment", the remaining tables in the hr schema are added to a 
destination database.

2. The dbs1.example.com database propagates these changes in the form of 
messages to a queue at dbs2.example.com.

3. At dbs2.example.com, DML changes to the jobs table are transformed into DML 
changes for the assignments table (which is a direct mapping of jobs) and then 
applied. Changes to other tables in the hr schema are not applied 
at dbs2.example.com.

4. Because the queue at dbs3.example.com receives changes from the queue at 
dbs2.example.com, these changes are propagated from dbs2.example.com to 
dbs3.example.com. This configuration is an example of directed networks.

5. The apply process at dbs3.example.com applies changes to the countries, 
locations, and regions tables.

6. Because dbs4.example.com, a Sybase database, receives changes from the queue 
at dbs2.example.com to the jobs table that originated at dbs1.example.com, these 
changes are applied remotely from dbs2.example.com using the 
dbs4.example.com database link through an Oracle Database Gateway. This configuration is an 
example of heterogeneous support.

2.2 Prerequisites

The following prerequisites must be completed before you begin the example in this 
chapter.

- Set the following initialization parameters to the values indicated for all databases 
in the environment:
  - GLOBAL_NAMES: This parameter must be set to TRUE at each database that is 
    participating in your Oracle Streams environment.
  - COMPATIBLE: This parameter must be set to 10.2.0 or higher.
  - STREAMS_POOL_SIZE: Optionally set this parameter to an appropriate value for 
    each database in the environment. This parameter specifies the size of the 
    Oracle Streams pool. The Oracle Streams pool stores messages in a buffered
queue and is used for internal communications during parallel capture and apply. When the MEMORY_TARGET, MEMORY_MAX_TARGET, or SGA_TARGET initialization parameter is set to a nonzero value, the Oracle Streams pool size is managed automatically.

**See Also:**

*Oracle Streams Replication Administrator's Guide* for information about other initialization parameters that are important in an Oracle Streams environment.

- Any database producing changes that will be captured must be running in ARCHIVELOG mode. In this example, changes are produced at dbs1.example.com, and so **dbs1.example.com must be running in ARCHIVELOG mode**.

**See Also:**

*Oracle Database Administrator's Guide* for information about running a database in ARCHIVELOG mode.

- Configure an Oracle Database Gateway on **dbs2.example.com** to communicate with the Sybase database **dbs4.example.com**.

**See Also:**

*Oracle Database Heterogeneous Connectivity User's Guide*

- At the Sybase database **dbs4.example.com**, set up the **hr user**.

**See Also:**

Your Sybase documentation for information about creating users and tables in your Sybase database.

- **Instantiate the hr.jobs table from the dbs1.example.com Oracle database at the dbs4.example.com Sybase database**.

**See Also:**

*Oracle Streams Replication Administrator's Guide*

- Configure your network and Oracle Net so that the following databases can communicate with each other:
  - **dbs1.example.com and dbs2.example.com**
  - **dbs2.example.com and dbs3.example.com**
Create an Oracle Streams administrator at each Oracle database in the replication environment. In this example, the databases are \texttt{dbs1.example.com}, \texttt{dbs2.example.com}, and \texttt{dbs3.example.com}. This example assumes that the user name of the Oracle Streams administrator is \texttt{strmadmin}.

### See Also:

*Oracle Database Net Services Administrator's Guide*

Create Queues and Database Links

Complete the following steps to create queues and database links for an Oracle Streams replication environment that includes three Oracle databases and one Sybase database:

1. Show Output and Spool Results
2. Create the \texttt{ANYDATA} Queue at \texttt{dbs1.example.com}
3. Create the Database Link at \texttt{dbs1.example.com}
4. Create the \texttt{ANYDATA} Queue at \texttt{dbs2.example.com}
5. Create the Database Links at \texttt{dbs2.example.com}
6. Create the \texttt{hr.assignments} Table at \texttt{dbs2.example.com}
7. Create the \texttt{ANYDATA} Queue at \texttt{dbs3.example.com}
8. Create the Database Links at \texttt{dbs2.example.com}
9. Drop All of the Tables in the \texttt{hr} Schema at \texttt{dbs3.example.com}
10. Check the Spool Results

#### Note:

If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line after this note to the next "END OF SCRIPT" line into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to all of the databases in the environment.
Show Output and Spool Results
Run **SET ECHO ON** and specify the spool file for the script. Check the spool file for errors after you run this script.

```sql
SET ECHO ON
SPOOL streams_setup_single.out
```

Create the ANYDATA Queue at **dbs1.example.com**
Connect as the Oracle Streams administrator at the database where you want to capture changes. In this example, that database is **dbs1.example.com**.

```sql
CONNECT strmadmin@dbs1.example.com
```

Run the **SET_UP_QUEUE** procedure to create a queue named **streams_queue** at **dbs1.example.com**. This queue will function as the ANYDATA queue by holding the captured changes that will be propagated to other databases.
Running the **SET_UP_QUEUE** procedure performs the following actions:

- Creates a queue table named **streams_queue_table**. This queue table is owned by the Oracle Streams administrator (**strmadmin**) and uses the default storage of this user.
- Creates a queue named **streams_queue** owned by the Oracle Streams administrator (**strmadmin**).
- Starts the queue.

```sql
EXEC DBMS_STREAMS_ADM.SET_UP_QUEUE();
```

Create the Database Link at **dbs1.example.com**
Create the database link from the database where changes are captured to the database where changes are propagated. In this example, the database where changes are captured is **dbs1.example.com**, and these changes are propagated to **dbs2.example.com**.

```sql
ACCEPT password PROMPT 'Enter password for user: ' HIDE
CREATE DATABASE LINK dbs2.example.com CONNECT TO strmadmin
    IDENTIFIED BY &password USING 'dbs2.example.com';
```

Create the ANYDATA Queue at **dbs2.example.com**
Connect as the Oracle Streams administrator at **dbs2.example.com**.
Run the `SET_UP_QUEUE` procedure to create a queue named `streams_queue` at `dbs2.example.com`. This queue will function as the ANYDATA queue by holding the changes that will be applied at this database and the changes that will be propagated to other databases.

Running the `SET_UP_QUEUE` procedure performs the following actions:

- Creates a queue table named `streams_queue_table`. This queue table is owned by the Oracle Streams administrator (`strmadmin`) and uses the default storage of this user.
- Creates a queue named `streams_queue` owned by the Oracle Streams administrator (`strmadmin`).
- Starts the queue.

```
EXEC DBMS_STREAMS_ADM.SET_UP_QUEUE();
```

Create the Database Links at `dbs2.example.com`

Create the database links to the databases where changes are propagated. In this example, database `dbs2.example.com` propagates changes to `dbs3.example.com`, which is another Oracle database, and to `dbs4.example.com`, which is a Sybase database. Notice that the database link to the Sybase database connects to the owner of the tables, not to the Oracle Streams administrator. This database link can connect to any user at `dbs4.example.com` that has privileges to change the `hr.jobs` table at that database.

```
CREATE DATABASE LINK dbs3.example.com CONNECT TO strmadmin IDENTIFIED BY &password USING 'dbs3.example.com';
CREATE DATABASE LINK dbs4.example.com CONNECT TO "hr" IDENTIFIED BY "&password" USING 'dbs4.example.com';
```

Create the `hr.assignments` Table at `dbs2.example.com`

This example illustrates a custom rule-based transformation in which changes to the `hr.jobs` table at `dbs1.example.com` are transformed into changes to the `hr.assignments`
table at dbs2.example.com. You must create the hr.assignments table on
dbs2.example.com for the transformation portion of this example to work properly.

Note:
Instead of using a custom rule-based transformation to change the name of
the table, you can use a RENAME_TABLE declarative rule-based transformation. See Oracle Streams Concepts and Administration.

Connect as hr at dbs2.example.com.

*/
CONNECT hr@dbs2.example.com
/*
Create the hr.assignments table in the dbs2.example.com database.

*/
CREATE TABLE hr.assignments AS SELECT * FROM hr.jobs;
ALTER TABLE hr.assignments ADD PRIMARY KEY (job_id);
/*
Create the ANYDATA Queue at dbs3.example.com
Connect as the Oracle Streams administrator at dbs3.example.com.

*/
CONNECT strmadmin@dbs3.example.com
/*
Run the SET_UP_QUEUE procedure to create a queue named streams_queue at
dbs3.example.com. This queue will function as the ANYDATA queue by holding the
changes that will be applied at this database.
Running the SET_UP_QUEUE procedure performs the following actions:
• Creates a queue table named streams_queue_table. This queue table is owned by
the Oracle Streams administrator (strmadmin) and uses the default storage of this
user.
• Creates a queue named streams_queue owned by the Oracle Streams
administrator (strmadmin).
• Starts the queue.

*/
EXEC DBMS_STREAMS_ADM.SET_UP_QUEUE();
/*
Create a Database Link at dbs3.example.com to dbs1.example.com

Create a database link from dbs3.example.com to dbs1.example.com. Later in this example, this database link is used for the instantiation of some of the database objects that were dropped in Step Create Queues and Database Links. This example uses the `DBMS_DATAPUMP` package to perform a network import of these database objects directly from the `dbs1.example.com` database. Because this example performs a network import, no dump file is required. Alternatively, you can perform an export at the source database `dbs1.example.com`, transfer the export dump file to the destination database `dbs3.example.com`, and then import the export dump file at the destination database. In this case, the database link created in this step is not required.

```
CREATE DATABASE LINK dbs1.example.com CONNECT TO strmadmin
    IDENTIFIED BY &password USING 'dbs1.example.com';
```

Drop All of the Tables in the hr Schema at dbs3.example.com

This example illustrates instantiating tables in the `hr` schema by importing them from `dbs1.example.com` into `dbs3.example.com` with Data Pump. You must delete these tables at `dbs3.example.com` for the instantiation portion of this example to work properly. Connect as `hr` at `dbs3.example.com`.

```
CONNECT hr@dbs3.example.com
```

Drop all tables in the `hr` schema in the `dbs3.example.com` database.

```
DROP TABLE hr.countries CASCADE CONSTRAINTS;
DROP TABLE hr.departments CASCADE CONSTRAINTS;
DROP TABLE hr.employees CASCADE CONSTRAINTS;
DROP TABLE hr.job_history CASCADE CONSTRAINTS;
DROP TABLE hr.jobs CASCADE CONSTRAINTS;
DROP TABLE hr.locations CASCADE CONSTRAINTS;
DROP TABLE hr.regions CASCADE CONSTRAINTS;
```

**Note:**

If you complete this step and drop all of the tables in the `hr` schema, then you should complete the remaining sections of this example to reinstantiate the `hr` schema at `dbs3.example.com`. If the `hr` schema does not exist in an Oracle database, then some examples in the Oracle documentation set can fail.

```
DROP TABLE hr.countries CASCADE CONSTRAINTS;
DROP TABLE hr.departments CASCADE CONSTRAINTS;
DROP TABLE hr.employees CASCADE CONSTRAINTS;
DROP TABLE hr.job_history CASCADE CONSTRAINTS;
DROP TABLE hr.jobs CASCADE CONSTRAINTS;
DROP TABLE hr.locations CASCADE CONSTRAINTS;
DROP TABLE hr.regions CASCADE CONSTRAINTS;
```

Check the Spool Results

Check the `streams_setup_single.out` spool file to ensure that all actions finished successfully after this script is completed.
2.4 Example Scripts for Sharing Data from One Database

This example illustrates two ways to accomplish the replication of the tables in the hr schema using Oracle Streams.

- "Simple Configuration for Sharing Data from a Single Database" demonstrates a simple way to configure the environment. This example uses the DBMS_STREAMS_ADM package to create a capture process, propagations, and apply processes, as well as the rule sets associated with them. Using the DBMS_STREAMS_ADM package is the simplest way to configure an Oracle Streams environment.

- "Flexible Configuration for Sharing Data from a Single Database" demonstrates a more flexible way to configure this environment. This example uses the DBMS_CAPTURE_ADM package to create a capture process, the DBMS_PROPAGATION_ADM package to create propagations, and the DBMS.Apply_ADM package to create apply processes. Also, this example uses the DBMS_RULES_ADM package to create and populate the rule sets associated with these capture processes, propagations, and apply processes. Using these packages, instead of the DBMS_STREAMS_ADM package, provides more configuration options and flexibility.

Note:

These examples illustrate two different ways to configure the same Oracle Streams environment. Therefore, you should run only one of the examples for a particular distributed database system. Otherwise, errors stating that objects already exist will result.

2.4.1 Simple Configuration for Sharing Data from a Single Database

Complete the following steps to specify the capture, propagation, and apply definitions using primarily the DBMS_STREAMS_ADM package.

1. Show Output and Spool Results
2. Configure Propagation at dbs1.example.com
3. Configure the Capture Process at dbs1.example.com
4. Set the Instantiation SCN for the Existing Tables at Other Databases
5. Instantiate the dbs1.example.com Tables at dbs3.example.com
6. Configure the Apply Process at dbs3.example.com
7. Specify hr as the Apply User for the Apply Process at dbs3.example.com
8. Grant the hr User Execute Privilege on the Apply Process Rule Set
9. Start the Apply Process at dbs3.example.com
10. Configure Propagation at dbs2.example.com
11. Create the Custom Rule-Based Transformation for Row LCRs at 
dbs2.example.com

12. Configure the Apply Process for Local Apply at dbs2.example.com

13. Specify hr as the Apply User for the Apply Process at dbs2.example.com

14. Grant the hr User Execute Privilege on the Apply Process Rule Set

15. Start the Apply Process at dbs2.example.com for Local Apply

16. Configure the Apply Process at dbs2.example.com for Apply at dbs4.example.com

17. Start the Apply Process at dbs2.example.com for Apply at dbs4.example.com

18. Start the Capture Process at dbs1.example.com

19. Check the Spool Results

---

Note:

If you are viewing this document online, then you can copy the text from the
"BEGINNING OF SCRIPT" line after this note to the next "END OF SCRIPT"
line into a text editor and then edit the text to create a script for your
environment. Run the script with SQL*Plus on a computer that can connect to
all of the databases in the environment.

/**************************** BEGINNING OF SCRIPT ***************************/

Show Output and Spool Results
Run SET ECHO ON and specify the spool file for the script. Check the spool file for errors
after you run this script.

*/

SET ECHO ON
SPOOL streams_share_schema1.out

/>

Configure Propagation at dbs1.example.com
Connect to dbs1.example.com as the strmadmin user.

*/

CONNECT strmadmin@dbs1.example.com

/>

Configure and schedule propagation of DML and DDL changes in the hr schema from
the queue at dbs1.example.com to the queue at dbs2.example.com.

*/

BEGIN
    DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES(
        schema_name => 'hr',
        ...
    );

*/

End of Script

---
BEGIN
    DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
        schema_name    => 'hr',
        streams_type   => 'capture',
        streams_name   => 'capture',
        queue_name     => 'strmadmin.streams_queue',
        include_dml    => TRUE,
        include_ddl    => TRUE,
        source_database => 'dbs1.example.com',
        inclusion_rule => TRUE,
        queue_to_queue => TRUE);
END;
/
/
/*

Configure the Capture Process at dbs1.example.com

Configure the capture process to capture changes to the entire hr schema at
dbs1.example.com. This step specifies that changes to the tables in the specified
schema are captured by the capture process and enqueued into the specified queue.
This step also prepares the hr schema for instantiation and enables supplemental
logging for any primary key, unique key, bitmap index, and foreign key columns in the
tables in this schema. Supplemental logging places additional information in the redo
log for changes made to tables. The apply process needs this extra information to
perform certain operations, such as unique row identification and conflict resolution.
Because dbs1.example.com is the only database where changes are captured in this
environment, it is the only database where you must specify supplemental logging for
the tables in the hr schema.

See Also:

Oracle Streams Replication Administrator's Guide
*/

BEGIN
    DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
        schema_name    => 'hr',
        streams_type   => 'capture',
        streams_name   => 'capture',
        queue_name     => 'strmadmin.streams_queue',
        include_dml    => TRUE,
        include_ddl    => TRUE,
        source_database => 'dbs1.example.com',
        inclusion_rule => TRUE,
        queue_to_queue => TRUE);
END;
/
/
/*

Set the Instantiation SCN for the Existing Tables at Other Databases

In this example, the hr.jobs table already exists at dbs2.example.com and
dbs4.example.com. At dbs2.example.com, this table is named assignments, but it has the
same shape and data as the jobs table at dbs1.example.com. Also, in this example,
dbs4.example.com is a Sybase database. All of the other tables in the Oracle Streams
environment are instantiated at the other destination databases using Data Pump
import.
Because the hr.jobs table already exists at dbs2.example.com and dbs4.example.com,
this example uses the GET_SYSTEM_CHANGE_NUMBER function in the DBMS_FLASHBACK

Chapter 2
Example Scripts for Sharing Data from One Database
package at dbs1.example.com to obtain the current SCN for the database. This SCN is used at dbs2.example.com to run the SET_TABLE_INSTANTIATION_SCN procedure in the DBMS_APPLY_ADM package. Running this procedure twice sets the instantiation SCN for the hr.jobs table at dbs2.example.com and dbs4.example.com. The SET_TABLE_INSTANTIATION_SCN procedure controls which LCRs for a table are ignored by an apply process and which LCRs for a table are applied by an apply process. If the commit SCN of an LCR for a table from a source database is less than or equal to the instantiation SCN for that table at a destination database, then the apply process at the destination database discards the LCR. Otherwise, the apply process applies the LCR.

In this example, both of the apply processes at dbs2.example.com will apply transactions to the hr.jobs table with SCNs that were committed after SCN obtained in this step.

Note:

This example assumes that the contents of the hr.jobs table at dbs1.example.com, dbs2.example.com (as hr.assignments), and dbs4.example.com are consistent when you complete this step. You might want to lock the table at each database while you complete this step to ensure consistency.

```
DECLARE
  iscn  NUMBER;         -- Variable to hold instantiation SCN value
BEGIN
  iscn := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER();
  DBMS_APPLY_ADM.SET_TABLE_INSTANTIATION_SCN@DBS2.EXAMPLE.COM(
    source_object_name    => 'hr.jobs',
    source_database_name  => 'dbs1.example.com',
    instantiation_scn     => iscn);
  DBMS_APPLY_ADM.SET_TABLE_INSTANTIATION_SCN@DBS2.EXAMPLE.COM(
    source_object_name    => 'hr.jobs',
    source_database_name  => 'dbs1.example.com',
    instantiation_scn     => iscn,
    apply_database_link   => 'dbs4.example.com');
END;
/*

Instantiate the dbs1.example.com Tables at dbs3.example.com
This example performs a network Data Pump import of the following tables:

- hr.countries
- hr.locations
- hr.regions

A network import means that Data Pump imports these tables from dbs1.example.com without using an export dump file.
```
See Also:

Oracle Database Utilities for information about performing an import

Connect to dbs3.example.com as the strmadmin user.

```
CONNECT strmadmin@dbs3.example.com
```

This example will do a table import using the DBMS_DATAPUMP package. For simplicity, exceptions from any of the API calls will not be trapped. However, Oracle recommends that you define exception handlers and call GET_STATUS to retrieve more detailed error information if a failure occurs. If you want to monitor the import, then query the DBA_DATAPUMP_JOBS data dictionary view at the import database.

```
SET SERVEROUTPUT ON
DECLARE
  h1        NUMBER;         -- Data Pump job handle
  sscn      NUMBER;         -- Variable to hold current source SCN
  job_state VARCHAR2(30);   -- To keep track of job state
  js        ku$_JobStatus;  -- The job status from GET_STATUS
  sts       ku$_Status;     -- The status object returned by GET_STATUS
  job_not_exist    exception;
  pragma exception_init(job_not_exist, -31626);
BEGIN
  -- Create a (user-named) Data Pump job to do a table-level import.
  h1 := DBMS_DATAPUMP.OPEN(  
    operation   => 'IMPORT',
    job_mode    => 'TABLE',
    remote_link => 'DBS1.EXAMPLE.COM',
    job_name    => 'dp_sing1');

  -- A metadata filter is used to specify the schema that owns the tables
  -- that will be imported.
  DBMS_DATAPUMP.METADATA_FILTER(  
    handle => h1,
    name   => 'SCHEMA_EXPR',
    value  => '=' 'HR');

  -- A metadata filter is used to specify the tables that will be imported.
  DBMS_DATAPUMP.METADATA_FILTER(  
    handle => h1,
    name   => 'NAME_EXPR',
    value  => 'IN(' 'COUNTRIES',' 'REGIONS',' 'LOCATIONS')');
  -- Get the current SCN of the source database, and set the FLASHBACK_SCN
  -- parameter to this value to ensure consistency between all of the
  -- objects included in the import.
  sscn := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER@dbs1.example.com();
  DBMS_DATAPUMP.SET_PARAMETER(  
    handle => h1,
    name   => 'FLASHBACK_SCN',
    value  => sscn);

  -- Start the job.
  DBMS_DATAPUMP.START_JOB(h1);
```
-- The import job should be running. In the following loop, the job
-- is monitored until it completes.
job_state := 'UNDEFINED';
BEGIN
  WHILE (job_state != 'COMPLETED') AND (job_state != 'STOPPED') LOOP
    sts:=DBMS_DATAPUMP.GET_STATUS(
      handle => h1,
      mask  => DBMS_DATAPUMP.KU$_STATUS_JOB_ERROR +
               DBMS_DATAPUMP.KU$_STATUS_JOB_STATUS +
               DBMS_DATAPUMP.KU$_STATUS_WIP,
      timeout => -1);
    js := sts.job_status;
    DBMS_LOCK.SLEEP(10);
    job_state := js.state;
  END LOOP;
  -- Gets an exception when job no longer exists
  EXCEPTION WHEN job_not_exist THEN
    DBMS_OUTPUT.PUT_LINE('Data Pump job has completed');
    DBMS_OUTPUT.PUT_LINE('Instantiation SCN: ' || sscn);
  END;
END;
/

Configure the Apply Process at dbs3.example.com
Connect to dbs3.example.com as the strmadmin user.

BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(  
    table_name      => 'hr.countries',
    streams_type    => 'apply',
    streams_name    => 'apply',
    queue_name      => 'strmadmin.streams_queue',
    include_dml     => TRUE,
    include_ddl     => TRUE,
    source_database => 'dbs1.example.com',
    inclusion_rule  => TRUE);
END;
/

BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(  
    table_name      => 'hr.locations',
    streams_type    => 'apply',
    streams_name    => 'apply',
    queue_name      => 'strmadmin.streams_queue',
    include_dml     => TRUE,
Specify hr as the Apply User for the Apply Process at dbs3.example.com
In this example, the hr user owns all of the database objects for which changes are applied by the apply process at this database. Therefore, hr already has the necessary privileges to change these database objects, and it is convenient to make hr the apply user.
When the apply process was created in the previous step, the Oracle Streams administrator strmadmin was specified as the apply user by default, because strmadmin ran the procedure that created the apply process. Instead of specifying hr as the apply user, you could retain strmadmin as the apply user, but then you must grant strmadmin privileges on all of the database objects for which changes are applied and privileges to execute all user procedures used by the apply process. In an environment where an apply process applies changes to database objects in multiple schemas, it might be more convenient to use the Oracle Streams administrator as the apply user.

See Also:
Oracle Streams Replication Administrator's Guide for more information about configuring an Oracle Streams administrator

Grant the hr User Execute Privilege on the Apply Process Rule Set
Because the hr user was specified as the apply user in the previous step, the hr user requires EXECUTE privilege on the positive rule set used by the apply process
DECLARE
    rs_name  VARCHAR2(64);  -- Variable to hold rule set name
BEGIN
    SELECT RULE_SET_OWNER||'.'||RULE_SET_NAME
    INTO rs_name
    FROM DBA_APPLY
    WHERE APPLY_NAME='APPLY';
    DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE(
        privilege   => SYS.DBMS_RULE_ADM.EXECUTE_ON_RULE_SET,
        object_name => rs_name,
        grantee     => 'hr');
END;
/

Start the Apply Process at dbs3.example.com
Set the disable_on_error parameter to N so that the apply process will not be disabled if it encounters an error, and start the apply process at dbs3.example.com.

BEGIN
    DBMS_APPLY_ADM.SET_PARAMETER(
        apply_name  => 'apply',
        parameter   => 'disable_on_error',
        value       => 'N');
END;
/

BEGIN
    DBMS_APPLY_ADM.START_APPLY(
        apply_name  => 'apply');
END;
/

Configure Propagation at dbs2.example.com
Connect to dbs2.example.com as the strmadmin user.

CONNECT strmadmin@dbs2.example.com
/

Configure and schedule propagation from the queue at dbs2.example.com to the queue at dbs3.example.com. You must specify this propagation for each table that will apply changes at dbs3.example.com. This configuration is an example of directed networks because the changes at dbs2.example.com originated at dbs1.example.com.

BEGIN
    DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES(
        table_name => 'hr.countries',
        /*
BEGIN
DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES(
  table_name => 'hr.locations',
  streams_name => 'dbs2_to_dbs3',
  source_queue_name => 'strmadmin.streams_queue',
  destination_queue_name => 'strmadmin.streams_queue@dbs3.example.com',
  include_dml => TRUE,
  include_ddl => TRUE,
  source_database => 'dbs1.example.com',
  inclusion_rule => TRUE);
END;
/

BEGIN
DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES(
  table_name => 'hr.regions',
  streams_name => 'dbs2_to_dbs3',
  source_queue_name => 'strmadmin.streams_queue',
  destination_queue_name => 'strmadmin.streams_queue@dbs3.example.com',
  include_dml => TRUE,
  include_ddl => TRUE,
  source_database => 'dbs1.example.com',
  inclusion_rule => TRUE);
END;
/

CREATE THE CUSTOM RULE-BASED TRANSFORMATION FOR ROW LCRS AT DBS2.EXAMPLE.COM
Connect to dbs2.example.com as the hr user.
*/

CONNECT hr@dbs2.example.com
/

Create the custom rule-based transformation function that transforms row changes resulting from DML statements to the jobs table from dbs1.example.com into row changes to the assignments table on dbs2.example.com. The following function transforms every row LCR for the jobs table into a row LCR for the assignments table.
If DDL changes were also applied to the assignments table, then another transformation would be required for the DDL LCRs. This transformation would need to change the object name and the DDL text.

```sql
CREATE OR REPLACE FUNCTION hr.to_assignments_trans_dml(
p_in_data in ANYDATA)
RETURN ANYDATA IS out_data SYS.LCR$ ROW_RECORD;
tc pls_integer;
BEGIN
  -- Typecast AnyData to LCR$ ROW_RECORD
  tc := p_in_data.GetObject(out_data);
  IF out_data.GET_OBJECT_NAME() = 'JOBS'
  THEN
    -- Transform the in_data into the out_data
    out_data.SET_OBJECT_NAME('ASSIGNMENTS');
    END IF;
  -- Convert to AnyData
  RETURN ANYDATA.ConvertObject(out_data);
END;
/

Configure the Apply Process for Local Apply at dbs2.example.com
Connect to dbs2.example.com as the strmadmin user.
*/

CONNECT strmadmin@dbs2.example.com
/*

Configure dbs2.example.com to apply changes to the assignments table. Remember that the assignments table receives changes from the jobs table at dbs1.example.com.
 */

DECLARE
  to_assignments_rulename_dml VARCHAR2(30);
dummy_rule VARCHAR2(30);
BEGIN
  -- DML changes to the jobs table from dbs1.example.com are applied
  -- to the assignments table. The to_assignments_rulename_dml variable
  -- is an out parameter in this call.
  DBMS_STREAMS_ADM ADD_TABLE_RULES(
    table_name => 'hr.jobs', -- jobs, not assignments, specified
    streams_type => 'apply',
    streams_name => 'apply_dbs2',
    queue_name => 'strmadmin.streams_queue',
    include_dml => TRUE,
    include_ddl => FALSE,
    source_database => 'dbs1.example.com',
    dml_rule_name => to_assignments_rulename_dml,
    /*
    
    Note:
    If DDL changes were also applied to the assignments table, then another transformation would be required for the DDL LCRs. This transformation would need to change the object name and the DDL text.
    */
```
-- Modify the rule for the hr.jobs table to use the transformation function.
DBMS_STREAMSADM.SET_RULE_TRANSFORM_FUNCTION(
    rule_name          => to_assignments_rule_name_dml,
    transform_function => 'hr.to_assignments_transform_dml');
END;
/

/*

Specify hr as the Apply User for the Apply Process at dbs2.example.com
In this example, the hr user owns all of the database objects for which changes are
applied by the apply process at this database. Therefore, hr already has the
necessary privileges to change these database objects, and it is convenient to make
hr the apply user.

When the apply process was created in the previous step, the Oracle Streams
administrator strmadmin was specified as the apply user by default, because strmadmin
ran the procedure that created the apply process. Instead of specifying hr as the apply
user, you could retain strmadmin as the apply user, but then you must grant strmadmin
privileges on all of the database objects for which changes are applied and privileges
to execute all user procedures used by the apply process. In an environment where
an apply process applies changes to database objects in multiple schemas, it might
be more convenient to use the Oracle Streams administrator as the apply user.

See Also:
Oracle Streams Replication Administrator's Guide for more information about
configuring an Oracle Streams administrator
*/
BEGIN
    DBMS_APPLY_ADM.ALTER_APPLY(
        apply_name => 'apply_dbs2',
        apply_user => 'hr');
END;
/

/*
Grant the hr User Execute Privilege on the Apply Process Rule Set
Because the hr user was specified as the apply user in the previous step, the hr user
requires EXECUTE privilege on the positive rule set used by the apply process
*/
DECLARE
    rs_name VARCHAR2(64);  -- Variable to hold rule set name
BEGIN
    SELECT RULE_SET_OWNER||'.'||RULE_SET_NAME
    INTO rs_name
    FROM DBA_APPLY
    WHERE APPLY_NAME='APPLY_DBS2';
    DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE(
privilege  => SYS.DBMS_RULE_ADM.EXECUTE_ON_RULE_SET,
object_name => rs_name,
grantee     => 'hr');
END;
/
/*

Start the Apply Process at dbs2.example.com for Local Apply
Set the disable_on_error parameter to n so that the apply process will not be disabled if it encounters an error, and start the apply process for local apply at dbs2.example.com.
*/
BEGIN
DBMS_APPLY_ADM.SET_PARAMETER(
  apply_name  => 'apply_dbs2',
  parameter   => 'disable_on_error',
  value       => 'N');
END;
/
BEGIN
DBMS_APPLY_ADM.START_APPLY(
  apply_name => 'apply_dbs2');
END;
/
/*

Configure the Apply Process at dbs2.example.com for Apply at dbs4.example.com
Configure the apply process for dbs4.example.com, which is a Sybase database. The dbs2.example.com database is acting as a gateway to dbs4.example.com. Therefore, the apply process for dbs4.example.com must be configured at dbs2.example.com. The apply process cannot apply DDL changes to non-Oracle databases. Therefore, the include_ddl parameter is set to FALSE when the ADD_TABLE_RULES procedure is run.
*/
BEGIN
DBMS_APPLY_ADM.CREATE_APPLY(
  queue_name          => 'strmadmin.streams_queue',
  apply_name          => 'apply_dbs4',
  apply_database_link => 'dbs4.example.com',
  apply_captured      => TRUE);
END;
/
BEGIN
DBMS_STREAMS_ADM.ADD_TABLE_RULES(
  table_name      => 'hr.jobs',
  streams_type    => 'apply',
  streams_name    => 'apply_dbs4',
  queue_name      => 'strmadmin.streams_queue',
  include_dml     => TRUE,
  include_ddl     => FALSE,
  source_database => 'dbs1.example.com',
  include_ddl     => FALSE,
  source_database => 'dbs1.example.com',
  include_ddl     => FALSE,
  source_database => 'dbs1.example.com',
  include_ddl     => FALSE,
  source_database => 'dbs1.example.com',
  include_ddl     => FALSE,
  source_database => 'dbs1.example.com',
  include_ddl     => FALSE,
  source_database => 'dbs1.example.com',
  include_ddl     => FALSE,
  source_database => 'dbs1.example.com',
  include_ddl     => FALSE,
  source_database => 'dbs1.example.com',
  include_ddl     => FALSE,
  source_database => 'dbs1.example.com',
  include_ddl     => FALSE,
  source_database => 'dbs1.example.com',
  include_ddl     => FALSE,
  source_database => 'dbs1.example.com',
  include_ddl     => FALSE,
  source_database => 'dbs1.example.com',
  include_ddl     => FALSE,
  source_database => 'dbs1.example.com',
  include_ddl     => FALSE,
  source_database => 'dbs1.example.com',
  include_ddl     => FALSE,
  source_database => 'dbs1.example.com',
  include_ddl     => FALSE,
  source_database => 'dbs1.example.com',
  include_ddl     => FALSE,
  source_database => 'dbs1.example.com',
  include_ddl     => FALSE,
  source_database => 'dbs1.example.com',
  include_ddl     => FALSE,
  source_database => 'dbs1.example.com',
  include_ddl     => FALSE,
  source_database => 'dbs1.example.com',
  include_ddl     => FALSE,
  source_database => 'dbs1.example.com',
  include_ddl     => FALSE,
  source_database => 'dbs1.example.com',
  include_ddl     => FALSE,
  source_database => 'dbs1.example.com',
  include_ddl     => FALSE,
  source_database => 'dbs1.example.com',
  include_ddl     => FALSE,
  source_database => 'dbs1.example.com',
  include_ddl     => FALSE,
  source_database => 'dbs1.example.com',
  include_ddl     => FALSE,
  source_database => 'dbs1.example.com',
  include_ddl     => FALSE,
  source_database => 'dbs1.example.com',
  include_ddl     => FALSE,
  source_database => 'dbs1.example.com',
  include_ddl     => FALSE,
  source_database => 'dbs1.example.com',
  include_ddl     => FALSE,
  source_database => 'dbs1.example.com',
  include_ddl     => FALSE,
  source_database => 'dbs1.example.com',
  include_ddl     => FALSE,
  source_database => 'dbs1.example.com',
  include_ddl     => FALSE,
  source_database => 'dbs1.example.com',
  include_ddl     => FALSE,
  source_database => 'dbs1.example.com',
  include_ddl     => FALSE,
Start the Apply Process at dbs2.example.com for Apply at dbs4.example.com
Set the disable_on_error parameter to n so that the apply process will not be disabled if it encounters an error, and start the remote apply for Sybase using database link dbs4.example.com.

```
BEGIN
  DBMS_APPLY_ADM.SET_PARAMETER(
    apply_name  => 'apply_dbs4',
    parameter   => 'disable_on_error',
    value       => 'N');
END;
/
```

Start the Capture Process at dbs1.example.com
Connect to dbs1.example.com as the strmadmin user.

```
CONNECT strmadmin@dbs1.example.com
```

Start the capture process at dbs1.example.com.

```
BEGIN
  DBMS_CAPTURE_ADM.START_CAPTURE(
    capture_name => 'capture');
END;
/
```

Check the Spool Results
Check the streams_share_schema1.out spool file to ensure that all actions finished successfully after this script is completed.

```
*/
SET ECHO OFF
SPOOL OFF
*/
You can now make DML and DDL changes to specific tables at dbs1.example.com and see these changes replicated to the other databases in the environment based on the rules you configured for the Oracle Streams processes and propagations in this environment.

See Also:

"Make DML and DDL Changes to Tables in the hr Schema" for examples of changes that are replicated in this environment

2.4.2 Flexible Configuration for Sharing Data from a Single Database

Complete the following steps to use a more flexible approach for specifying the capture, propagation, and apply definitions. This approach does not use the DBMS_STREAMS_ADM package. Instead, it uses the following packages:

- The DBMS_CAPTURE_ADM package to configure capture processes
- The DBMS_PROPAGATION_ADM package to configure propagations
- The DBMS_APPLY_ADM package to configure apply processes
- The DBMS_RULES_ADM package to specify capture process, propagation, and apply process rules and rule sets

Note:

Neither the ALL_STREAMS_TABLE_RULES nor the DBA_STREAMS_TABLE_RULES data dictionary view is populated by the rules created in this example. To view the rules created in this example, you can query the ALL_STREAMS_RULES or DBA_STREAMS_RULES data dictionary view.

This example includes the following steps:

1. Show Output and Spool Results
2. Configure Propagation at dbs1.example.com
3. Configure the Capture Process at dbs1.example.com
4. Prepare the hr Schema at dbs1.example.com for Instantiation
5. Set the Instantiation SCN for the Existing Tables at Other Databases
6. Instantiate the dbs1.example.com Tables at dbs3.example.com
7. Configure the Apply Process at dbs3.example.com
8. Grant the hr User Execute Privilege on the Apply Process Rule Set
9. Start the Apply Process at dbs3.example.com
10. Configure Propagation at dbs3.example.com
11. Create the Custom Rule-Based Transformation for Row LCRs at dbs2.example.com
12. Configure the Apply Process for Local Apply at dbs2.example.com
13. Grant the hr User Execute Privilege on the Apply Process Rule Set
14. Start the Apply Process at dbs2.example.com for Local Apply
15. Configure the Apply Process at dbs2.example.com for Apply at dbs4.example.com
16. Start the Apply Process at dbs2.example.com for Apply at dbs4.example.com
17. Start the Capture Process at dbs1.example.com
18. Check the Spool Process at dbs1.example.com

Note:
If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line after this note to the next "END OF SCRIPT" line into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to all of the databases in the environment.

/************************* BEGINNING OF SCRIPT ***************************/

Show Output and Spool Results
Run SET ECHO ON and specify the spool file for the script. Check the spool file for errors after you run this script.

*/

SET ECHO ON
SPOOL streams_share_schema2.out

*/

Configure Propagation at dbs1.example.com
Connect to dbs1.example.com as the strmadmin user.

*/

CONNECT strmadmin@dbs1.example.com

*/

Configure and schedule propagation from the queue at dbs1.example.com to the queue at dbs2.example.com. This configuration specifies that the propagation propagates all changes to the hr schema. You have the option of omitting the rule set specification, but then everything in the queue will be propagated, which might not be desired if, in the future, multiple capture processes will use the streams_queue.

*/

BEGIN
-- Create the rule set
DBMS_RULE_ADM.CREATE_RULE_SET(
  rule_set_name => 'strmadmin.propagation_dbs1_rules',
  evaluation_context => 'SYS.STREAMS$_EVALUATION_CONTEXT');
-- Create rules for all modifications to the hr schema
DBMS_RULE_ADM.CREATE_RULE(
    rule_name => 'strmadmin.all_hr_dml',
    condition => ' :dml.get_object_owner() = '"HR"' AND ' ||
        ' :dml.is_null_tag() = '"Y"' AND ' ||
        ' :dml.get_source_database_name() = '"DBS1.EXAMPLE.COM" ');

DBMS_RULE_ADM.CREATE_RULE(
    rule_name => 'strmadmin.all_hr_ddl',
    condition => ' (:ddl.get_object_owner() = '"HR"' OR ' ||
        ' :ddl.get_base_table_owner() = '"HR") AND ' ||
        ' :ddl.is_null_tag() = '"Y"' AND ' ||
        ' :ddl.get_source_database_name() = '"DBS1.EXAMPLE.COM" ');

-- Add rules to rule set
DBMS_RULE_ADM.ADD_RULE(
    rule_name => 'strmadmin.all_hr_dml',
    rule_set_name => 'strmadmin.propagation_dbs1_rules');

DBMS_RULE_ADM.ADD_RULE(
    rule_name => 'strmadmin.all_hr_ddl',
    rule_set_name => 'strmadmin.propagation_dbs1_rules');

-- Create a propagation that uses the rule set as its positive rule set
DBMS_PROPAGATION_ADM.CREATE_PROPAGATION(
    propagation_name => 'dbs1_to_dbs2',
    source_queue => 'strmadmin.streams_queue',
    destination_queue => 'strmadmin.streams_queue',
    destination_dblink => 'dbs2.example.com',
    rule_set_name => 'strmadmin.propagation_dbs1_rules');

END;
/

Configure the Capture Process at dbs1.example.com

Create a capture process and rules to capture the entire hr schema at dbs1.example.com.
*/

BEGIN

-- Create the rule set
DBMS_RULE_ADM.CREATE_RULE_SET(
    rule_set_name => 'strmadmin.demo_rules',
    evaluation_context => 'SYS.STREAMS$_EVALUATION_CONTEXT');

-- Create rules that specify the entire hr schema
DBMS_RULE_ADM.CREATE_RULE(
    rule_name => 'strmadmin.schema_hr_dml',
    condition => ' :dml.get_object_owner() = '"HR"' AND ' ||
        ' :dml.is_null_tag() = '"Y"' AND ' ||
        ' :dml.get_source_database_name() = '"DBS1.EXAMPLE.COM" ');

DBMS_RULE_ADM.CREATE_RULE(
    rule_name => 'strmadmin.schema_hr_ddl',
    condition => ' (:ddl.get_object_owner() = '"HR"' OR ' ||
        ' :ddl.get_base_table_owner() = '"HR") AND ' ||
        ' :ddl.is_null_tag() = '"Y"' AND ' ||
        ' :ddl.get_source_database_name() = '"DBS1.EXAMPLE.COM" ');

-- Add the rules to the rule set
DBMS_RULE_ADM.ADD_RULE(
    rule_name => 'strmadmin.schema_hr_dml',
    rule_set_name => 'strmadmin.demo_rules');

DBMS_RULE_ADM.ADD_RULE(
    rule_name => 'strmadmin.schema_hr_ddl',
    rule_set_name => 'strmadmin.demo_rules');
```
rule_name      => 'strmadmin.schema_hr_ddl',
rule_set_name  => 'strmadmin.demo_rules');
-- Create a capture process that uses the rule set as its positive rule set
DBMS_CAPTURE_ADM.CREATE_CAPTURE(
  queue_name    => 'strmadmin.streams_queue',
capture_name   => 'capture',
rule_set_name  => 'strmadmin.demo_rules');
END;
/
/*

Prepare the hr Schema at dbs1.example.com for Instantiation
While still connected as the Oracle Streams administrator at dbs1.example.com,
prepare the hr schema at dbs1.example.com for instantiation at dbs3.example.com. This
step marks the lowest SCN of the tables in the schema for instantiation. SCNs
subsequent to the lowest SCN can be used for instantiation.
This step also enables supplemental logging for any primary key, unique key, bitmap
index, and foreign key columns in the tables in the hr schema. Supplemental logging
places additional information in the redo log for changes made to tables. The apply
process needs this extra information to perform certain operations, such as unique
row identification and conflict resolution. Because dbs1.example.com is the only
database where changes are captured in this environment, it is the only database
where you must specify supplemental logging for the tables in the hr schema.

Note:
This step is not required in the "Simple Configuration for Sharing Data from a
Single Database". In that example, when the ADD_SCHEMA_RULES procedure in the
DBMS_STREAMS_ADM package is run in Step Configure the Capture Process at
dbs1.example.com Configure the Capture Process at dbs1.example.com, the
PREPARE_SCHEMA_INTEGRATION procedure in the DBMS_CAPTURE_ADM package is
run automatically for the hr schema.

See Also:
Oracle Streams Replication Administrator's Guide

*/
BEGIN
  DBMS_CAPTURE_ADM.PREPARE_SCHEMA_INTEGRATION{
    schema_name        => 'hr',
    supplemental_logging => 'keys'};
END;
/
/*

Set the Instantiation SCN for the Existing Tables at Other Databases
In this example, the hr.jobs table already exists at dbs2.example.com and
dbs4.example.com. At dbs2.example.com, this table is named assignments, but it has the
same shape and data as the jobs table at dbs1.example.com. Also, in this example,
**Note:**

This example assumes that the contents of the `hr.jobs` table at `dbs1.example.com`, `dbs2.example.com` (as `hr.assignments`), and `dbs4.example.com` are consistent when you complete this step. You might want to lock the table at each database while you complete this step to ensure consistency.

```
/*
DECLARE
  iscn  NUMBER;         -- Variable to hold instantiation SCN value
BEGIN
  iscn := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER();
  DBMS_APPLY_ADM.SET_TABLE_INSTANTIATION_SCN@DBS2.EXAMPLE.COM(
    source_object_name    => 'hr.jobs',
    source_database_name  => 'dbs1.example.com',
    instantiation_scn     => iscn);
  DBMS_APPLY_ADM.SET_TABLE_INSTANTIATION_SCN@DBS2.EXAMPLE.COM(
    source_object_name    => 'hr.jobs',
    source_database_name  => 'dbs1.example.com',
    instantiation_scn     => iscn,
    apply_database_link   => 'dbs4.example.com');
END;
/

/*

**Instantiate the dbs1.example.com Tables at dbs3.example.com**

This example performs a network Data Pump import of the following tables:

- `hr.countries`
- `hr.locations`
- `hr.regions`
A network import means that Data Pump imports these tables from \texttt{dbs1.example.com} without using an export dump file.

See Also:

Oracle Database Utilities for information about performing an import

Connect to \texttt{dbs3.example.com} as the \texttt{strmadmin} user.

```
CONNECT strmadmin@dbs3.example.com
```

This example will do a table import using the \texttt{DBMS_DATAPUMP} package. For simplicity, exceptions from any of the API calls will not be trapped. However, Oracle recommends that you define exception handlers and call \texttt{GET_STATUS} to retrieve more detailed error information if a failure occurs. If you want to monitor the import, then query the \texttt{DBA_DATAPUMP_JOBS} data dictionary view at the import database.

```
SET SERVEROUTPUT ON
DECLARE
  h1        NUMBER;         -- Data Pump job handle
  sscn      NUMBER;         -- Variable to hold current source SCN
  job_state VARCHAR2(30);   -- To keep track of job state
  js        ku$_JobStatus;  -- The job status from \texttt{GET_STATUS}
  sts       ku$_Status;     -- The status object returned by \texttt{GET_STATUS}
  job_not_exist    exception;
  pragma exception_init(job_not_exist, -31626);
BEGIN
  -- Create a (user-named) Data Pump job to do a table-level import.
  h1 := DBMS_DATAPUMP.OPEN(
    operation   => 'IMPORT',
    job_mode    => 'TABLE',
    remote_link => 'DDBS1.EXAMPLE.COM',
    job_name    => 'dp_sing2');
  -- A metadata filter is used to specify the schema that owns the tables
  -- that will be imported.
  DBMS_DATAPUMP.METADATA_FILTER(
    handle     => h1,
    name       => 'SCHEMA_EXPR',
    value      => '=' 'HR');
  -- A metadata filter is used to specify the tables that will be imported.
  DBMS_DATAPUMP.METADATA_FILTER(
    handle     => h1,
    name       => 'NAME_EXPR',
    value      => 'IN(' 'COUNTRIES', 'REGIONS', 'LOCATIONS')');
  -- Get the current SCN of the source database, and set the \texttt{FLASHBACK_SCN}
  -- parameter to this value to ensure consistency between all of the
  -- objects included in the import.
  sscn := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER@dbs1.example.com();
  DBMS_DATAPUMP.SET_PARAMETER(
    handle => h1,
```

```
name  => 'FLASHBACK_SCN',
value  => sscn);
-- Start the job.
DBMS_DATAPUMP.START_JOB(h1);
-- The import job should be running. In the following loop, the job
-- is monitored until it completes.
job_state := 'UNDEFINED';
BEGIN
  WHILE (job_state != 'COMPLETED') AND (job_state != 'STOPPED') LOOP
    sts:=DBMS_DATAPUMP.GET_STATUS(
       handle  => h1,
       mask    => DBMS_DATAPUMP.KU$_STATUS_JOB_ERROR +
                   DBMS_DATAPUMP.KU$_STATUS_JOB_STATUS +
                   DBMS_DATAPUMP.KU$_STATUS_WIP,
       timeout => -1);
    js := sts.job_status;
    DBMS_LOCK.SLEEP(10);
    job_state := js.state;
  END LOOP;
-- Gets an exception when job no longer exists
EXCEPTION WHEN job_not_exist THEN
  DBMS_OUTPUT.PUT_LINE('Data Pump job has completed');
  DBMS_OUTPUT.PUT_LINE('Instantiation SCN: ' || sscn);
END;
END;
/

Chapter 2
Example Scripts for Sharing Data from One Database

Configure the Apply Process at dbs3.example.com
Configure dbs3.example.com to apply DML and DDL changes to the countries table, locations table, and regions table.

BEGIN
-- Create the rule set
DBMS_RULE_ADM.CREATE_RULE_SET(
   rule_set_name => 'stradmin.apply_rules',
   evaluation_context => 'SYS.STREAMS$_EVALUATION_CONTEXT');
-- Rules for hr.countries
DBMS_RULE_ADM.CREATE_RULE(
   rule_name => 'stradmin.all_countries_dml',
   condition => ' :dml.get_object_owner() = 'HR' AND ' ||
                ' :dml.get_object_name() = 'COUNTRIES' AND ' ||
                ' :dml.is_null_tag() = 'Y' AND ' ||
                ' :dml.get_source_database_name() = ''DBS1.EXAMPLE.COM'' ');
DBMS_RULE_ADM.CREATE_RULE(
   rule_name => 'stradmin.all_countries_ddl',
   condition => ' :ddl.get_object_owner() = 'HR' OR ' ||
                ' :ddl.get_base_table_owner() = ''HR'' AND ' ||
                ' :ddl.get_base_owner() = 'HR' AND ' ||
-- Rules for hr.locations
DBMS_RULE_ADM.CREATE_RULE(
    rule_name    => 'strmadmin.all_locations_dml',
    condition    => ':dml.get_object_owner() = 'HR'' AND ' ||
                    ':dml.get_object_name() = 'LOCATIONS' AND ' ||
                    ':dml.is_null_tag() = 'Y' AND ' ||
                    ':dml.get_source_database_name() = 'DBS1.EXAMPLE.COM'' ');

DBMS_RULE_ADM.CREATE_RULE(
    rule_name    => 'strmadmin.all_locations_ddl',
    condition    => '(:ddl.get_object_owner() = 'HR'' OR ' ||
                    ':ddl.get_base_table_owner() = 'HR'')) AND ' ||
                    ':ddl.get_object_name() = 'LOCATIONS' AND ' ||
                    ':ddl.is_null_tag() = 'Y' AND ' ||
                    ':ddl.get_source_database_name() = 'DBS1.EXAMPLE.COM'' ');

-- Rules for hr.regions
DBMS_RULE_ADM.CREATE_RULE(
    rule_name    => 'strmadmin.all_regions_dml',
    condition    => ':dml.get_object_owner() = 'HR'' AND ' ||
                    ':dml.get_object_name() = 'REGIONS' AND ' ||
                    ':dml.is_null_tag() = 'Y' AND ' ||
                    ':dml.get_source_database_name() = 'DBS1.EXAMPLE.COM'' ');

DBMS_RULE_ADM.CREATE_RULE(
    rule_name    => 'strmadmin.all_regions_ddl',
    condition    => '(:ddl.get_object_owner() = 'HR'' OR ' ||
                    ':ddl.get_base_table_owner() = 'HR'')) AND ' ||
                    ':ddi.get_object_name() = 'REGIONS' AND ' ||
                    ':ddi.is_null_tag() = 'Y' AND ' ||
                    ':ddi.get_source_database_name() = 'DBS1.EXAMPLE.COM'' ');

-- Add rules to rule set
DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'strmadmin.all_countries_dml',
    rule_set_name  => 'strmadmin.apply_rules');

DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'strmadmin.all_countries_ddl',
    rule_set_name  => 'strmadmin.apply_rules');

DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'strmadmin.all_locations_dml',
    rule_set_name  => 'strmadmin.apply_rules');

DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'strmadmin.all_locations_ddl',
    rule_set_name  => 'strmadmin.apply_rules');

DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'strmadmin.all_regions_dml',
    rule_set_name  => 'strmadmin.apply_rules');

DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'strmadmin.all_regions_ddl',
    rule_set_name  => 'strmadmin.apply_rules');

-- Create an apply process that uses the rule set as its positive rule set
DBMS_APPLY_ADM.CREATE_APPLY(
    queue_name      => 'strmadmin.streams_queue',
    apply_name      => 'apply',
    rule_set_name   => 'strmadmin.apply_rules',
    apply_user      => 'hr',
    apply_captured  => TRUE,
    source_database => 'dbs1.example.com');

END;
Grant the hr User Execute Privilege on the Apply Process Rule Set
Because the hr user was specified as the apply user in the previous step, the hr user requires EXECUTE privilege on the positive rule set used by the apply process

```
BEGIN
DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE(
    privilege   => SYS.DBMS_RULE_ADM.EXECUTE_ON_RULE_SET,
    object_name => 'strmadmin.apply_rules',
    grantee     => 'hr');
END;
/
```

Start the Apply Process at dbs3.example.com
Set the disable_on_error parameter to n so that the apply process will not be disabled if it encounters an error, and start the apply process at dbs3.example.com.

```
BEGIN
DBMS_APPLY_ADM.SET_PARAMETER(
    apply_name  => 'apply',
    parameter   => 'disable_on_error',
    value       => 'N');
END;
/
BEGIN
DBMS_APPLY_ADM.START_APPLY(
    apply_name  => 'apply');
END;
/
```

Configure Propagation at dbs2.example.com
Connect to dbs2.example.com as the strmadmin user.

```
CONNECT strmadmin@dbs2.example.com
```

```
Configure and schedule propagation from the queue at dbs2.example.com to the queue at dbs3.example.com. This configuration is an example of directed networks because the changes at dbs2.example.com originated at dbs1.example.com.
```

```
BEGIN
    -- Create the rule set
```
DBMS_RULE_ADM.CREATE_RULE_SET(
    rule_set_name => 'strmadmin.propagation_dbs3_rules',
    evaluation_context => 'SYS.STREAMS$EVALUATION_CONTEXT');

-- Create rules for all modifications to the countries table
DBMS_RULE_ADM.CREATE_RULE(
    rule_name => 'strmadmin.all_countries_dml',
    condition => ' :dml.get_object_owner() = 'HR' AND ' ||
               ' :dml.get_object_name() = 'COUNTRIES' AND ' ||
               ' :dml.is_null_tag() = 'Y' AND ' ||
               ' :dml.get_source_database_name() = 'DBS1.EXAMPLE.COM' ');

DBMS_RULE_ADM.CREATE_RULE(
    rule_name => 'strmadmin.all_countries_ddl',
    condition => ' (:ddl.get_object_owner() = 'HR' OR ' ||
               ' :ddl.get_base_table_owner() = 'HR') AND ' ||
               ' :ddl.get_object_name() = 'COUNTRIES' AND ' ||
               ' :ddl.is_null_tag() = 'Y' AND ' ||
               ' :ddl.get_source_database_name() = 'DBS1.EXAMPLE.COM' ');

-- Create rules for all modifications to the locations table
DBMS_RULE_ADM.CREATE_RULE(
    rule_name => 'strmadmin.all_locations_dml',
    condition => ' :dml.get_object_owner() = 'HR' AND ' ||
               ' :dml.get_object_name() = 'LOCATIONS' AND ' ||
               ' :dml.is_null_tag() = 'Y' AND ' ||
               ' :dml.get_source_database_name() = 'DBS1.EXAMPLE.COM' ');

DBMS_RULE_ADM.CREATE_RULE(
    rule_name => 'strmadmin.all_locations_ddl',
    condition => ' (:ddl.get_object_owner() = 'HR' OR ' ||
               ' :ddl.get_base_table_owner() = 'HR') AND ' ||
               ' :ddl.get_object_name() = 'LOCATIONS' AND ' ||
               ' :ddl.is_null_tag() = 'Y' AND ' ||
               ' :ddl.get_source_database_name() = 'DBS1.EXAMPLE.COM' ');

-- Create rules for all modifications to the regions table
DBMS_RULE_ADM.CREATE_RULE(
    rule_name => 'strmadmin.all_regions_dml',
    condition => ' :dml.get_object_owner() = 'HR' AND ' ||
               ' :dml.get_object_name() = 'REGIONS' AND ' ||
               ' :dml.is_null_tag() = 'Y' AND ' ||
               ' :dml.get_source_database_name() = 'DBS1.EXAMPLE.COM' ');

DBMS_RULE_ADM.CREATE_RULE(
    rule_name => 'strmadmin.all_regions_ddl',
    condition => ' (:ddl.get_object_owner() = 'HR' OR ' ||
               ' :ddl.get_base_table_owner() = 'HR') AND ' ||
               ' :ddl.get_object_name() = 'REGIONS' AND ' ||
               ' :ddl.is_null_tag() = 'Y' AND ' ||
               ' :ddl.get_source_database_name() = 'DBS1.EXAMPLE.COM' ');

-- Add rules to rule set
DBMS_RULE_ADM.ADD_RULE(
    rule_name => 'strmadmin.all_countries_dml',
    rule_set_name => 'strmadmin.propagation_dbs3_rules');

DBMS_RULE_ADM.ADD_RULE(
    rule_name => 'strmadmin.all_countries_ddl',
    rule_set_name => 'strmadmin.propagation_dbs3_rules');

DBMS_RULE_ADM.ADD_RULE(
    rule_name => 'strmadmin.all_locations_dml',
    rule_set_name => 'strmadmin.propagation_dbs3_rules');

DBMS_RULE_ADM.ADD_RULE(
    rule_name => 'strmadmin.all_locations_ddl',
    rule_set_name => 'strmadmin.propagation_dbs3_rules');
rule_name      => 'strmadmin.all_regions_dml',
rule_set_name  => 'strmadmin.propagation_dbs3_rules');
DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'strmadmin.all_regions_ddl',
    rule_set_name  => 'strmadmin.propagation_dbs3_rules');
-- Create a propagation that uses the rule set as its positive rule set
DBMS_PROPAGATION_ADM.CREATE_PROPAGATION(
    propagation_name    => 'dbs2_to_dbs3',
    source_queue        => 'strmadmin.streams_queue',
    destination_queue   => 'strmadmin.streams_queue',
    destination_dblink  => 'dbs3.example.com',
    rule_set_name       => 'strmadmin.propagation_dbs3_rules');
END;
/

 Create the Custom Rule-Based Transformation for Row LCRs at 
dbs2.example.com

Connect to dbs2.example.com as the hr user.

/*
CREATE OR REPLACE FUNCTION hr.to_assignments_trans_dml(
p_in_data in ANYDATA)
RETURN ANYDATA IS out_data SYS.LCR$_ROW_RECORD;
tc   pls_integer;
BEGIN
    -- Typecast AnyData to LCR$_ROW_RECORD
    tc := p_in_data.GetObject(out_data);
    IF out_data.GET_OBJECT_NAME() = 'JOBS'
    THEN
        -- Transform the in_data into the out_data
        out_data.SET_OBJECT_NAME('ASSIGNMENTS');
        END IF;
    -- Convert to AnyData
    RETURN ANYDATA.ConvertObject(out_data);
END;
Configure the Apply Process for Local Apply at dbs2.example.com

Connect to dbs2.example.com as the strmadmin user.

CONNECT strmadmin@dbs2.example.com

Configure dbs2.example.com to apply changes to the local assignments table. Remember that the assignments table receives changes from the jobs table at dbs1.example.com. This step specifies a custom rule-based transformation without using the SET_RULE_TRANSFORM_FUNCTION procedure in the DBMS_STREAMS_ADM package. Instead, a name-value pair is added manually to the action context of the rule. The name-value pair specifies STREAMS$_TRANSFORM_FUNCTION for the name and hr.to_assignments_trans_dml for the value.

DECLARE
  action_ctx_dml       SYS.RE$NV_LIST;
  action_ctx_ddl       SYS.RE$NV_LIST;
  ac_name              VARCHAR2(30) := 'STREAMS$_TRANSFORM_FUNCTION';
BEGIN
  -- Specify the name-value pair in the action context
  action_ctx_dml := SYS.RE$NV_LIST(SYS.RE$NV_ARRAY());
  action_ctx_dml.ADD_PAIR(
    ac_name,
    ANYDATA.CONVERTVARCHAR2('hr.to_assignments_trans_dml'));
  -- Create the rule set strmadmin.apply_rules
  DBMS_RULE_ADM.CREATE_RULE_SET(
    rule_set_name       => 'strmadmin.apply_rules',
    evaluation_context  => 'SYS.STREAMS$_EVALUATION_CONTEXT');
  -- Create a rule that transforms all DML changes to the jobs table into
  -- DML changes for assignments table
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name       => 'strmadmin.all_jobs_dml',
    condition       => :
dml.get_object_owner() = 'HR' AND '||
                      ':dml.get_object_name() = 'JOBS' AND '||
                      ':dml.is_null_tag() = 'Y' AND '||
                      ':dml.get_source_database_name() = 'DBS1.EXAMPLE.COM',
    action_context  => action_ctx_dml);
  -- Add the rule to the rule set
  DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'strmadmin.all_jobs_dml',
    rule_set_name  => 'strmadmin.apply_rules');
  -- Create an apply process that uses the rule set as its positive rule set
  DBMS_APPLY_ADM.CREATE_APPLY(
    queue_name      => 'strmadmin.streams_queue',
    apply_name      => 'apply_dbs2',
    rule_set_name   => 'strmadmin.apply_rules',
    apply_user      => 'hr',
    apply_captured => TRUE,
    source_database => 'dbs1.example.com');
END;
Grant the hr User Execute Privilege on the Apply Process Rule Set
Because the hr user was specified as the apply user in the previous step, the hr user requires EXECUTE privilege on the positive rule set used by the apply process

```
BEGIN
	DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE(
	    privilege => SYS.DBMS_RULE_ADM.EXECUTE_ON_RULE_SET,
	    object_name => 'strmadmin.apply_rules',
	    grantee => 'hr');
END;
/
/*

Start the Apply Process at dbs2.example.com for Local Apply
Set the disable_on_error parameter to N so that the apply process will not be disabled if it encounters an error, and start the apply process for local apply at dbs2.example.com.

```
```
BEGIN
	DBMS_APPLY_ADM.SET_PARAMETER(
	    apply_name => 'apply_dbs2',
	    parameter => 'disable_on_error',
	    value => 'N');
END;
/
```
```
BEGIN
	DBMS_APPLY_ADM.START_APPLY(
	    apply_name => 'apply_dbs2');
END;
/
/*

Configure the Apply Process at dbs2.example.com for Apply at dbs4.example.com
Configure dbs2.example.com to apply DML changes to the jobs table at dbs4.example.com, which is a Sybase database. Remember that these changes originated at dbs1.example.com.

```
BEGIN
	-- Create the rule set
	DBMS_RULE_ADM.CREATE_RULE_SET(
	    rule_set_name => 'strmadmin.apply_dbs4_rules',
	    evaluation_context => 'SYS.STREAMS$_EVALUATION_CONTEXT');
	-- Create rule strmadmin.all_jobs_remote for all modifications
	-- to the jobs table
	DBMS_RULE_ADM.CREATE_RULE(
```
```
rule_name      => 'strmadmin.all_jobs_remote',
condition      => ' :dml.get_object_owner() = ''HR'' AND ' ||
                 ' :dml.get_object_name() = ''JOBS'' AND ' ||
                 ' :dml.is_null_tag() = ''Y'' AND ' ||
                 ' :dml.get_source_database_name() = ''DBS1.EXAMPLE.COM'' ');
-- Add the rule to the rule set
DBMS_RULE_ADM.ADD_RULE(
  rule_name      => 'strmadmin.all_jobs_remote',
  rule_set_name  => 'strmadmin.apply_dbs4_rules');
-- Create an apply process that uses the rule set as its positive rule set
DBMS_APPLY_ADM.CREATE_APPLY(
  queue_name          => 'strmadmin.streams_queue',
  apply_name          => 'apply_dbs4',
  rule_set_name       => 'strmadmin.apply_dbs4_rules',
  apply_database_link => 'dbs4.example.com',
  apply_captured      => TRUE,
  source_database     => 'dbs1.example.com');
END;
/

/*
Start the Apply Process at dbs2.example.com for Apply at dbs4.example.com
Set the disable_on_error parameter to n so that the apply process will not be disabled
if it encounters an error, and start the remote apply for Sybase using database link
dbs4.example.com.
*/
BEGIN
  DBMS_APPLY_ADM.SET_PARAMETER(
    apply_name  => 'apply_dbs4',
    parameter   => 'disable_on_error',
    value       => 'N');
END;
/

BEGIN
  DBMS_APPLY_ADM.START_APPLY(
    apply_name => 'apply_dbs4');
END;
/

/*
Start the Capture Process at dbs1.example.com
Connect to dbs1.example.com as the strmadmin user.
*/
CONNECT strmadmin@dbs1.example.com
/
Start the capture process at dbs1.example.com.
*/
BEGIN
```
DBMS_CAPTURE_ADM.START_CAPTURE(
    capture_name  => 'capture');
END;
/
/

/*
Check the Spool Results
Check the streams_share_schema2.out Spool file to ensure that all actions finished successfully after this script is completed.
*/

SET ECHO OFF
SPOOL OFF
/
/*
You can now make DML and DDL changes to specific tables at dbs1.example.com and see these changes replicated to the other databases in the environment based on the rules you configured for the Oracle Streams processes and propagations in this environment.

See Also:
"Make DML and DDL Changes to Tables in the hr Schema" for examples of changes that are replicated in this environment

/**************************** END OF SCRIPT ******************************/

2.5 Make DML and DDL Changes to Tables in the hr Schema

After completing either of the examples described in "Example Scripts for Sharing Data from One Database", you can make DML and DDL changes to the tables in the hr schema at the dbs1.example.com database. These changes will be replicated to the other databases in the environment based on the rules you configured for Oracle Streams processes and propagations. You can check the other databases to see that the changes have been replicated.

For example, complete the following steps to make DML changes to the hr.jobs and hr.locations tables at dbs1.example.com. You can also make a DDL change to the hr.locations table at dbs1.example.com.

After you make these changes, you can query the hr.assignments table at dbs2.example.com to see that the DML change you made to this table at dbs1.example.com has been replicated. Remember that a custom rule-based transformation configured for the apply process at dbs2.example.com transforms DML changes to the hr.jobs table into DML changes to the hr.assignments table. You can also query the hr.locations table at dbs3.example.com to see that the DML and DDL changes you made to this table at dbs1.example.com have been replicated.
Make DML and DDL Changes to Tables in the hr Schema

Make the following changes:

CONNECT hr@dbs1.example.com
Enter password: password

UPDATE hr.jobs SET max_salary=10000 WHERE job_id='MK_REP';
COMMIT;

INSERT INTO hr.locations VALUES(
    3300, '521 Ralston Avenue', '94002', 'Belmont', 'CA', 'US');
COMMIT;

ALTER TABLE hr.locations RENAME COLUMN state_province TO state_or_province;

Query the hr.assignments Table at dbs2.example.com

After some time passes to allow for capture, propagation, and apply of the changes performed the previous step, run the following query to confirm that the UPDATE change made to the hr.jobs table at dbs1.example.com has been applied to the hr.assignments table at dbs2.example.com.

CONNECT hr@dbs2.example.com
Enter password: password

SELECT max_salary FROM hr.assignments WHERE job_id='MK_REP';

You should see 10000 for the value of the max_salary.

Query and Describe the hr.locations Table at dbs3.example.com

Run the following query to confirm that the INSERT change made to the hr.locations table at dbs1.example.com has been applied at dbs3.example.com.

CONNECT hr@dbs3.example.com
Enter password: password

SELECT * FROM hr.locations WHERE location_id=3300;

You should see the row inserted into the hr.locations table at dbs1.example.com in the previous step.
Next, describe the hr.locations table at to confirm that the ALTER TABLE change was propagated and applied correctly.

DESC hr.locations

The fifth column in the table should be state_or_province.

2.6 Add Objects to an Existing Oracle Streams Replication Environment

This example extends the Oracle Streams environment configured in the previous sections by adding replicated objects to an existing database. To complete this example, you must have completed the tasks in one of the previous examples in this chapter.

This example will add the following tables to the hr schema in the dbs3.example.com database:

• departments
• employees
• job_history
• jobs

When you complete this example, Oracle Streams processes changes to these tables with the following series of actions:

1. The capture process captures changes at dbs1.example.com and enqueues them at dbs1.example.com.
2. A propagation propagates changes from the queue at dbs1.example.com to the queue at dbs2.example.com.
3. A propagation propagates changes from the queue at dbs2.example.com to the queue at dbs3.example.com.
4. The apply process at dbs3.example.com applies the changes at dbs3.example.com.

When you complete this example, the hr schema at the dbs3.example.com database will have all of its original tables, because the countries, locations, and regions tables were instantiated at dbs3.example.com in the previous section.

Figure 2-2 provides an overview of the environment with the added tables.
Complete the following steps to replicate these tables to the `dbs3.example.com` database.

1. Show Output and Spool Results
2. Stop the Apply Process at `dbs3.example.com`
3. Configure the Apply Process for the Added Tables at `dbs3.example.com`
4. Specify the Table Propagation Rules for the Added Tables at `dbs2.example.com`
5. Prepare the Four Added Tables for Instantiation at `dbs1.example.com`
6. Instantiate the `dbs1.example.com` Tables at `dbs3.example.com`
7. Start the Apply Process at `dbs3.example.com`
8. Check the Spool Results
Note:

If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line after this note to the next "END OF SCRIPT" line into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to all of the databases in the environment.

/************************* BEGINNING OF SCRIPT ***************************/

Show Output and Spool Results
Run SET ECHO ON and specify the spool file for the script. Check the spool file for errors after you run this script.

*/

SET ECHO ON
SPOOL streams_addobjs.out

/ *

Stop the Apply Process at dbs3.example.com
Until you finish adding objects to dbs3.example.com, you must ensure that the apply process that will apply changes for the added objects does not try to apply changes for these objects. You can do this by stopping the capture process at the source database. Or, you can do this by stopping propagation of changes from dbs2.example.com to dbs3.example.com. Yet another alternative is to stop the apply process at dbs3.example.com. This example stops the apply process at dbs3.example.com.

Connect to dbs3.example.com as the strmadmin user.

*/

CONNECT strmadmin@dbs3.example.com

/ *

Stop the apply process at dbs3.example.com.

*/

BEGIN
DBMS_APPLY_ADM.STOP_APPLY(
    apply_name => 'apply');
END;
/

/ *

Configure the Apply Process for the Added Tables at dbs3.example.com
Configure the apply process at dbs3.example.com to apply changes to the tables you are adding.

*/
BEGIN
DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name => 'hr.departments',
    streams_type => 'apply',
    streams_name => 'apply',
    queue_name => 'strmadmin.streams_queue',
    include_dml => TRUE,
    include_ddl => TRUE,
    source_database => 'dbs1.example.com',
    inclusion_rule => TRUE);
END;
/

BEGIN
DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name => 'hr.employees',
    streams_type => 'apply',
    streams_name => 'apply',
    queue_name => 'strmadmin.streams_queue',
    include_dml => TRUE,
    include_ddl => TRUE,
    source_database => 'dbs1.example.com',
    inclusion_rule => TRUE);
END;
/

BEGIN
DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name => 'hr.job_history',
    streams_type => 'apply',
    streams_name => 'apply',
    queue_name => 'strmadmin.streams_queue',
    include_dml => TRUE,
    include_ddl => TRUE,
    source_database => 'dbs1.example.com',
    inclusion_rule => TRUE);
END;
/

BEGIN
DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name => 'hr.jobs',
    streams_type => 'apply',
    streams_name => 'apply',
    queue_name => 'strmadmin.streams_queue',
    include_dml => TRUE,
    include_ddl => TRUE,
    source_database => 'dbs1.example.com',
    inclusion_rule => TRUE);
END;
/

/*
Specify the Table Propagation Rules for the Added Tables at dbs2.example.com
Connect to dbs2.example.com as the strmadmin user.
*/
CONNECT strmadmin@dbs2.example.com

/*
 * Add the tables to the rules for propagation from the queue at dbs2.example.com to the queue at dbs3.example.com.
 */

BEGIN
DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES(
    table_name => 'hr.departments',
    streams_name => 'dbs2_to_dbs3',
    source_queue_name => 'strmadmin.streams_queue',
    destination_queue_name => 'strmadmin.streams_queue@dbs3.example.com',
    include_dml => TRUE,
    include_ddl => TRUE,
    source_database => 'dbs1.example.com',
    inclusion_rule => TRUE);
END;
/

BEGIN
DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES(
    table_name => 'hr.employees',
    streams_name => 'dbs2_to_dbs3',
    source_queue_name => 'strmadmin.streams_queue',
    destination_queue_name => 'strmadmin.streams_queue@dbs3.example.com',
    include_dml => TRUE,
    include_ddl => TRUE,
    source_database => 'dbs1.example.com',
    inclusion_rule => TRUE);
END;
/

BEGIN
DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES(
    table_name => 'hr.job_history',
    streams_name => 'dbs2_to_dbs3',
    source_queue_name => 'strmadmin.streams_queue',
    destination_queue_name => 'strmadmin.streams_queue@dbs3.example.com',
    include_dml => TRUE,
    include_ddl => TRUE,
    source_database => 'dbs1.example.com',
    inclusion_rule => TRUE);
END;
/

BEGIN
DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES(
    table_name => 'hr.jobs',
    streams_name => 'dbs2_to_dbs3',
    source_queue_name => 'strmadmin.streams_queue',
    destination_queue_name => 'strmadmin.streams_queue@dbs3.example.com',
    include_dml => TRUE,
    include_ddl => TRUE,
    source_database => 'dbs1.example.com',
    inclusion_rule => TRUE);
END;
Prepare the Four Added Tables for Instantiation at dbs1.example.com

Connect to dbs1.example.com as the strmadmin user.

```sql
CONNECT strmadmin@dbs1.example.com
```

Prepare the tables for instantiation. These tables will be instantiated at dbs3.example.com. This step marks the lowest SCN of the tables for instantiation. SCNs subsequent to the lowest SCN can be used for instantiation. Also, this preparation is necessary so that the Oracle Streams data dictionary for the relevant propagations and the apply process at dbs3.example.com contain information about these tables.

**Note:**

When the `PREPARE_TABLE_INSTANTIATION` procedure is run in this step, the `supplemental_logging` parameter is not specified. Therefore, the default value (keys) is used for this parameter. Supplemental logging already was enabled for any primary key, unique key, bitmap index, and foreign key columns in these tables in Step 3.

**See Also:**

*Oracle Streams Replication Administrator's Guide*
DBMS_CAPTURE_ADM.PREPARE_TABLE_INSTANTIATION(
    table_name  => 'hr.jobs');
END;
/
/*

Instantiate the dbs1.example.com Tables at dbs3.example.com
This example performs a network Data Pump import of the following tables:

*  hr.departments
*  hr.employees
*  hr.job_history
*  hr.jobs

A network import means that Data Pump imports these tables from dbs1.example.com without using an export dump file.

See Also:
Oracle Database Utilities for information about performing an import

Connect to dbs3.example.com as the strmadmin user.
*/
CONNECT strmadmin@dbs3.example.com
/

This example will do a table import using the DBMS_DATAPUMP package. For simplicity, exceptions from any of the API calls will not be trapped. However, Oracle recommends that you define exception handlers and call GET_STATUS to retrieve more detailed error information if a failure occurs. If you want to monitor the import, then query the DBA_DATAPUMP_JOBS data dictionary view at the import database.
*/

SET SERVEROUTPUT ON
DECLARE
  h1        NUMBER;         -- Data Pump job handle
  sscn      NUMBER;         -- Variable to hold current source SCN
  job_state VARCHAR2(30);   -- To keep track of job state
  js        ku$_JobStatus;  -- The job status from GET_STATUS
  sts       ku$_Status;     -- The status object returned by GET_STATUS
  job_not_exist    exception;
  pragma exception_init(job_not_exist, -31626);
BEGIN
  -- Create a (user-named) Data Pump job to do a table-level import.
  h1 := DBMS_DATAPUMP.OPEN(
    operation => 'IMPORT',
    job_mode => 'TABLE',
    remote_link => 'DBS1.EXAMPLE.COM',
    job_name => 'dp_sing3');
  -- A metadata filter is used to specify the schema that owns the tables
-- that will be imported.
DBMS_DATAPUMP.METADATA_FILTER(
    handle => h1,
    name => 'SCHEMA_EXPR',
    value => '='|HR'||);

-- A metadata filter is used to specify the tables that will be imported.
DBMS_DATAPUMP.METADATA_FILTER(
    handle => h1,
    name => 'NAME_EXPR',
    value => 'IN('DEPARTMENTS', 'EMPLOYEES',
             'JOB_HISTORY', 'JOBS')');

-- Get the current SCN of the source database, and set the FLASHBACK_SCN
-- parameter to this value to ensure consistency between all of the
-- objects included in the import.
sscn := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER@dbs1.example.com();
DBMS_DATAPUMP.SET_PARAMETER(
    handle => h1,
    name => 'FLASHBACK_SCN',
    value => sscn);

-- Start the job.
DBMS_DATAPUMP.START_JOB(h1);

-- The import job should be running. In the following loop, the job
-- is monitored until it completes.
job_state := 'UNDEFINED';
BEGIN
    WHILE (job_state != 'COMPLETED') AND (job_state != 'STOPPED') LOOP
        sts:=DBMS_DATAPUMP.GET_STATUS(
            handle => h1,
            mask => DBMS_DATAPUMP.KU$_STATUS_JOB_ERROR +
                    DBMS_DATAPUMP.KU$_STATUS_JOB_STATUS +
                    DBMS_DATAPUMP.KU$_STATUS_WIP,
            timeout => -1);
        js := sts.job_status;
        DBMS_LOCK.SLEEP(10);
        job_state := js.state;
    END LOOP;

    -- Gets an exception when job no longer exists
    EXCEPTION WHEN job_not_exist THEN
        DBMS_OUTPUT.PUT_LINE('Data Pump job has completed');
        DBMS_OUTPUT.PUT_LINE('Instantiation SCN: ' || sscn);
    END;
END;
/

/*/ 
Start the Apply Process at dbs3.example.com
Start the apply process at dbs3.example.com. This apply process was stopped in Step
Stop the Apply Process at dbs3.example.com.
Connect to dbs3.example.com as the strmadmin user.
*/

CONNECT strmadmin@dbs3.example.com

Start the apply process at dbs3.example.com.
2.7 Make a DML Change to the hr.employees Table

After completing the examples described in the "Add Objects to an Existing Oracle Streams Replication Environment" section, you can make DML and DDL changes to the tables in the hr schema at the dbs1.example.com database. These changes will be replicated to dbs3.example.com. You can check these tables at dbs3.example.com to see that the changes have been replicated.

For example, complete the following steps to make a DML change to the hr.employees table at dbs1.example.com. Next, query the hr.employees table at dbs3.example.com to see that the change has been replicated.

**Make a DML Change to the hr.employees Table**

Make the following change:

```
CONNECT hr@dbs1.example.com
Enter password: password

UPDATE hr.employees SET job_id='ST_MAN' WHERE employee_id=143;
COMMIT;
```

**Query the hr.employees Table at dbs3.example.com**

After some time passes to allow for capture, propagation, and apply of the change performed in the previous step, run the following query to confirm that the UPDATE change made to the hr.employees table at dbs1.example.com has been applied to the hr.employees table at dbs3.example.com.

```
CONNECT hr@dbs3.example.com
Enter password: password

SELECT job_id FROM hr.employees WHERE employee_id=143;
```

You should see ST_MAN for the value of the job_id.
Add a Database to an Existing Oracle Streams Replication Environment

This example extends the Oracle Streams environment configured in the previous sections by adding an additional database to the existing configuration. In this example, an existing Oracle database named `dbs5.example.com` is added to receive changes to the entire `hr` schema from the queue at `dbs2.example.com`.

Figure 2-3 provides an overview of the environment with the added database.

To complete this example, you must meet the following prerequisites:
The `dbs5.example.com` database must exist.

The `dbs2.example.com` and `dbs5.example.com` databases must be able to communicate with each other through Oracle Net.

The `dbs5.example.com` and `dbs1.example.com` databases must be able to communicate with each other through Oracle Net (for optional Data Pump network instantiation).

You must have completed the tasks in the previous examples in this chapter.

The "Prerequisites" must be met if you want the entire Oracle Streams environment to work properly.

This examples creates a new user to function as the Oracle Streams administrator (strmadmin) at the `dbs5.example.com` database and prompts you for the tablespace you want to use for this user's data. Before you start this example, either create a new tablespace or identify an existing tablespace for the Oracle Streams administrator to use at the `dbs5.example.com` database. The Oracle Streams administrator should not use the `SYSTEM` tablespace.

Complete the following steps to add `dbs5.example.com` to the Oracle Streams environment.

1. Show Output and Spool Results
2. Drop All of the Tables in the hr Schema at `dbs5.example.com`
3. Set Up Users at `dbs5.example.com`
4. Create the ANYDATA Queue at `dbs5.example.com`
5. Create a Database Link at `dbs5.example.com` to `dbs1.example.com`
6. Configure the Apply Process at `dbs5.example.com`
7. Specify hr as the Apply User for the Apply Process at `dbs5.example.com`
8. Grant the hr User Execute Privilege on the Apply Process Rule Set
9. Create the Database Link Between `dbs2.example.com` and `dbs5.example.com`
10. Configure Propagation Between `dbs2.example.com` and `dbs5.example.com`
11. Prepare the hr Schema for Instantiation at `dbs1.example.com`
12. Instantiate the `dbs1.example.com` Tables at `dbs5.example.com`
13. Start the Apply Process at `dbs5.example.com`
14. Check the Spool Results

**Note:**

If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line after this note to the next "END OF SCRIPT" line into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to all of the databases in the environment.
Show Output and Spool Results
Run SET ECHO ON and specify the spool file for the script. Check the spool file for errors after you run this script.

/*
SET ECHO ON
SPOOL streams_adddb.out
/

Drop All of the Tables in the hr Schema at dbs5.example.com
This example illustrates instantiating the tables in the hr schema by importing them from dbs1.example.com into dbs5.example.com using Data Pump. You must delete these tables at dbs5.example.com for the instantiation portion of this example to work properly.
Connect as hr at dbs5.example.com.

/*
CONNECT hr@dbs5.example.com
/

Drop all tables in the hr schema in the dbs5.example.com database.

Note:
If you complete this step and drop all of the tables in the hr schema, then you should complete the remaining sections of this example to reinstantiate the hr schema at dbs5.example.com. If the hr schema does not exist in an Oracle database, then some examples in the Oracle documentation set can fail.

/*
DROP TABLE hr.countries CASCADE CONSTRAINTS;
DROP TABLE hr.departments CASCADE CONSTRAINTS;
DROP TABLE hr.employees CASCADE CONSTRAINTS;
DROP TABLE hr.job_history CASCADE CONSTRAINTS;
DROP TABLE hr.jobs CASCADE CONSTRAINTS;
DROP TABLE hr.locations CASCADE CONSTRAINTS;
DROP TABLE hr.regions CASCADE CONSTRAINTS;
/

Set Up Users at dbs5.example.com
Connect to dbs5.example.com as SYSTEM user.

/*
CONNECT system@dbs5.example.com
/

Create the Oracle Streams administrator named strmadmin and grant this user the necessary privileges. These privileges enable the user to manage queues, execute
subprograms in packages related to Oracle Streams, create rule sets, create rules, and monitor the Oracle Streams environment by querying data dictionary views and queue tables. You can choose a different name for this user.

Note:
The ACCEPT command must appear on a single line in the script.

See Also:
Oracle Streams Replication Administrator's Guide for more information about configuring an Oracle Streams administrator

*/

ACCEPT password PROMPT 'Enter password for user: ' HIDE
GRANT DBA TO strmadmin IDENTIFIED BY &password;

ACCEPT streams_tbs PROMPT 'Enter Oracle Streams administrator tablespace on dbs5.example.com: ' HIDE
ALTER USER strmadmin DEFAULT TABLESPACE &streams_tbs
QUOTA UNLIMITED ON &streams_tbs;

Create the ANYDATA Queue at dbs5.example.com
Connect as the Oracle Streams administrator at the database you are adding. In this example, that database is dbs5.example.com.

*/

CONNECT strmadmin@dbs5.example.com

/*

Run the SET_UP_QUEUE procedure to create a queue named streams_queue at dbs5.example.com. This queue will function as the ANYDATA queue by holding the changes that will be applied at this database. Running the SET_UP_QUEUE procedure performs the following actions:

• Creates a queue table named streams_queue_table. This queue table is owned by the Oracle Streams administrator (strmadmin) and uses the default storage of this user.

• Creates a queue named streams_queue owned by the Oracle Streams administrator (strmadmin).

• Starts the queue.

*/*
EXEC DBMSSTREAMSADM.SET_UP_QUEUE();

/*

Create a Database Link at dbs5.example.com to dbs1.example.com

Create a database link from dbs5.example.com to dbs1.example.com. Later in this example, this database link is used for the instantiation of the database objects that were dropped in Step Drop All of the Tables in the hr Schema at dbs5.example.com. This example uses the DBMS_DATAPUMP package to perform a network import of these database objects directly from the dbs1.example.com database. Because this example performs a network import, no dump file is required. Alternatively, you can perform an export at the source database dbs1.example.com, transfer the export dump file to the destination database dbs5.example.com, and then import the export dump file at the destination database. In this case, the database link created in this step is not required.

*/

CREATE DATABASE LINK dbs1.example.com CONNECT TO strmadmin
    IDENTIFIED BY &password USING 'dbs1.example.com';

/*

Configure the Apply Process at dbs5.example.com

While still connected as the Oracle Streams administrator at dbs5.example.com, configure the apply process to apply changes to the hr schema.

*/

BEGIN
    DBMSSTREAMSADM.ADD_SCHEMA_RULES(
        schema_name => 'hr',
        streams_type => 'apply',
        streams_name => 'apply',
        queue_name => 'strmadmin.streams_queue',
        include_dml => TRUE,
        include_ddl => TRUE,
        source_database => 'dbs1.example.com',
        inclusion_rule => TRUE);
END;
/

/*

Specify hr as the Apply User for the Apply Process at dbs5.example.com

In this example, the hr user owns all of the database objects for which changes are applied by the apply process at this database. Therefore, hr already has the necessary privileges to change these database objects, and it is convenient to make hr the apply user.

When the apply process was created in the previous step, the Oracle Streams administrator strmadmin was specified as the apply user by default, because strmadmin ran the procedure that created the apply process. Instead of specifying hr as the apply user, you could retain strmadmin as the apply user, but then you must grant strmadmin privileges on all of the database objects for which changes are applied and privileges to execute all user procedures used by the apply process. In an environment where an apply process applies changes to database objects in multiple schemas, it might be more convenient to use the Oracle Streams administrator as the apply user.

*/
See Also:

Oracle Streams Replication Administrator's Guide for more information about configuring an Oracle Streams administrator

/*
BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name => 'apply',
    apply_user => 'hr');
END;
/
/*

Grant the hr User Execute Privilege on the Apply Process Rule Set
Because the hr user was specified as the apply user in the previous step, the hr user requires EXECUTE privilege on the positive rule set used by the apply process

*/
DECLARE
  rs_name  VARCHAR2(64);   -- Variable to hold rule set name
BEGIN
  SELECT RULE_SET_OWNER||'.'||RULE_SET_NAME
  INTO rs_name
  FROM DBA_APPLY
  WHERE APPLY_NAME='APPLY';
  DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE(
    privilege   => SYS.DBMS_RULE_ADM.EXECUTE_ON_RULE_SET,
    object_name => rs_name,
    grantee     => 'hr');
END;
/
/*

Create the Database Link Between dbs2.example.com and dbs5.example.com
Connect to dbs2.example.com as the strmadmin user.

*/
CONNECT strmadmin@dbs2.example.com
/
/*

Create the database links to the databases where changes are propagated. In this example, database dbs2.example.com propagates changes to dbs5.example.com.

*/
CREATE DATABASE LINK dbs5.example.com CONNECT TO strmadmin
  IDENTIFIED BY &password USING 'dbs5.example.com';
/*
Configure Propagation Between dbs2.example.com and dbs5.example.com
While still connected as the Oracle Streams administrator at dbs2.example.com, configure and schedule propagation from the queue at dbs2.example.com to the queue at dbs5.example.com. Remember, changes to the hr schema originated at dbs1.example.com.

```sql
BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES(
    schema_name => 'hr',
    streams_name => 'dbs2_to_dbs5',
    source_queue_name => 'strmadmin.streams_queue',
    destination_queue_name => 'strmadmin.streams_queue@dbs5.example.com',
    include_dml => TRUE,
    include_ddl => TRUE,
    source_database => 'dbs1.example.com',
    inclusion_rule => TRUE,
    queue_to_queue => TRUE);
END;
/
```

Prepare the hr Schema for Instantiation at dbs1.example.com
Connect to dbs1.example.com as the strmadmin user.

```sql
BEGIN
  DBMS_CAPTURE_ADM.PREPARE_SCHEMA_INSTANTIATION(
    schema_name => 'hr',
    supplemental_logging => 'keys');
END;
/
```

Instantiate the dbs1.example.com Tables at dbs5.example.com
This example performs a network Data Pump import of the following tables:

See Also:
Oracle Streams Replication Administrator's Guide
A network import means that Data Pump imports these tables from `dbs1.example.com` without using an export dump file.

**See Also:**

*Oracle Database Utilities* for information about performing an import

Connect to `dbs5.example.com` as the `strmadmin` user.

```sql
CONNECT strmadmin@dbs5.example.com
```

This example will do a table import using the `DBMS_DATAPUMP` package. For simplicity, exceptions from any of the API calls will not be trapped. However, Oracle recommends that you define exception handlers and call `GET_STATUS` to retrieve more detailed error information if a failure occurs. If you want to monitor the import, then query the `DBA_DATAPUMP_JOBS` data dictionary view at the import database.

```sql
SET SERVEROUTPUT ON
DECLARE
    h1        NUMBER;         -- Data Pump job handle
    sscn      NUMBER;         -- Variable to hold current source SCN
    job_state VARCHAR2(30);   -- To keep track of job state
    js        ku$_JobStatus;  -- The job status from GET_STATUS
    sts       ku$_Status;     -- The status object returned by GET_STATUS
    job_not_exist    exception;
    pragma exception_init(job_not_exist, -31626);
BEGIN
    -- Create a (user-named) Data Pump job to do a table-level import.
    h1 := DBMS_DATAPUMP.OPEN(
        operation => 'IMPORT',
        job_mode => 'TABLE',
        remote_link => 'DBS1.EXAMPLE.COM',
        job_name => 'dp_sing4');
    -- A metadata filter is used to specify the schema that owns the tables
    -- that will be imported.
    DBMS_DATAPUMP.METADATA_FILTER(
        handle => h1,
        name => 'SCHEMA_EXPR',
```
value => 'HR');
-- Get the current SCN of the source database, and set the FLASHBACK_SCN
-- parameter to this value to ensure consistency between all of the
-- objects included in the import.
sscn := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER@dbs1.example.com();
DBMS_DATAPUMP.SET_PARAMETER(
    handle => h1,
    name => 'FLASHBACK_SCN',
    value => sscn);
-- Start the job.
DBMS_DATAPUMP.START_JOB(h1);
-- The import job should be running. In the following loop, the job
-- is monitored until it completes.
job_state := 'UNDEFINED';
BEGIN
    WHILE (job_state != 'COMPLETED') AND (job_state != 'STOPPED') LOOP
        sts := DBMS_DATAPUMP.GET_STATUS(
            handle => h1,
            mask => DBMS_DATAPUMP.KU$STATUS_JOB_ERROR +
                DBMS_DATAPUMP.KU$STATUS_JOB_STATUS +
                DBMS_DATAPUMP.KU$STATUS_WIP,
            timeout => -1);
        js := sts.job_status;
        DBMS_LOCK.SLEEP(10);
        job_state := js.state;
    END LOOP;
    -- Gets an exception when job no longer exists
    EXCEPTION WHEN job_not_exist THEN
        DBMS_OUTPUT.PUT_LINE('Data Pump job has completed');
        DBMS_OUTPUT.PUT_LINE('Instantiation SCN: ' || sscn);
    END;
END;
/

Start the Apply Process at dbs5.example.com
Connect as the Oracle Streams administrator at dbs5.example.com.
*/
CONNECT strmadmin@dbs5.example.com
/*
Set the disable_on_error parameter to N so that the apply process will not be disabled
if it encounters an error, and start apply process at dbs5.example.com.
*/
BEGIN
    DBMS_APPLY_ADM.SET_PARAMETER(
        apply_name => 'apply',
        parameter => 'disable_on_error',
        value => 'N');
END;
/
BEGIN
2.9 Make a DML Change to the hr.departments Table

After completing the examples described in the "Add a Database to an Existing Oracle Streams Replication Environment" section, you can make DML and DDL changes to the tables in the hr schema at the dbs1.example.com database. These changes will be replicated to dbs5.example.com. You can check these tables at dbs5.example.com to see that the changes have been replicated.

For example, complete the following steps to make a DML change to the hr.departments table at dbs1.example.com. Next, query the hr.departments table at dbs5.example.com to see that the change has been replicated.

Make a DML Change to the hr.departments Table

Make the following change:

CONNECT hr@dbs1.example.com
Enter password: password

UPDATE hr.departments SET location_id=2400 WHERE department_id=270;
COMMIT;

Query the hr.departments Table at dbs5.example.com

After some time passes to allow for capture, propagation, and apply of the change performed in the previous step, run the following query to confirm that the UPDATE change made to the hr.departments table at dbs1.example.com has been applied to the hr.departments table at dbs5.example.com.

CONNECT hr@dbs5.example.com
Enter password: password

SELECT location_id FROM hr.departments WHERE department_id=270;

You should see 2400 for the value of the location_id.
This chapter illustrates an example of an n-way replication environment that can be constructed using Oracle Streams.

This chapter contains these topics:

- Overview of the N-Way Replication Example
- Prerequisites
- Create the hrmult Schema at the mult1.example.com Database
- Create Queues and Database Links
- Example Script for Configuring N-Way Replication
- Make DML and DDL Changes to Tables in the hrmult Schema

### 3.1 Overview of the N-Way Replication Example

This example illustrates using Oracle Streams to replicate data for a schema among three Oracle databases. DML and DDL changes made to tables in the hrmult Schema are captured at all databases in the environment and propagated to each of the other databases in the environment.

This type of environment is called an n-way replication environment. An n-way replication environment is a type of multiple-source replication environment because more than one source database captures and replicates changes.

**Figure 3-1** provides an overview of the environment.
As illustrated in Figure 3-1, all of the databases will contain the `hrmult` schema when the example is complete. However, at the beginning of the example, the `hrmult` schema exists only at `mult1.example.com`. During the example, you instantiate the `hrmult` schema at `mult2.example.com` and `mult3.example.com`.

In this example, Oracle Streams is used to perform the following series of actions:

1. After instantiation, the capture process at each database captures DML and DDL changes for all of the tables in the `hrmult` schema and enqueues them into a local queue.
2. Propagations at each database propagate these changes to all of the other databases in the environment.
3. The apply processes at each database apply changes in the `hrmult` schema received from the other databases in the environment.

This example avoids sending changes back to their source database by using the default apply tag for the apply processes. When you create an apply process, the changes applied by the apply process have redo entries with a tag of '00' (double zero) by default. These changes are not recaptured because, by default, rules created by the `DBMS_STREAMS_ADM` package have an `is_null_tag()='Y'` condition by default, and this condition ensures that each capture process captures a change in a redo entry only if the tag for the redo entry is `NULL`.

**See Also:**

- *Oracle Streams Replication Administrator's Guide* for more information about n-way replication environments
- *Oracle Streams Replication Administrator's Guide* for more information about tags

### 3.2 Prerequisites

The following prerequisites must be completed before you begin the example in this chapter.

- Set the following initialization parameters to the values indicated at each database in the Oracle Streams environment:
  - **GLOBAL_NAMES**: This parameter must be set to `TRUE`. Ensure that the global names of the databases are `mult1.example.com`, `mult2.example.com`, and `mult3.example.com`.
  - **COMPATIBLE**: This parameter must be set to `10.2.0` or higher.
  - Ensure that the `PROCESSES` and `SESSIONS` initialization parameters are set high enough for all of the Oracle Streams clients used in this example. This example configures one capture process, two propagations, and two apply processes at each database.
  - **STREAMS_POOL_SIZE**: Optionally set this parameter to an appropriate value for each database in the environment. This parameter specifies the size of the Oracle Streams pool. The Oracle Streams pool stores messages in a buffered queue and is used for internal communications during parallel capture and apply. When the `MEMORY_TARGET`, `MEMORY_MAX_TARGET`, or `SGA_TARGET` initialization parameter is set to a nonzero value, the Oracle Streams pool size is managed automatically.

**Note:**

You might need to modify other initialization parameter settings for this example to run properly.
3.3 Create the hrmult Schema at the mult1.example.com Database

For the purposes of this example, create a new schema named hrmult at the mult1.example.com database. The n-way environment will replicate this new schema.

Complete the following steps to use Data Pump export/import to create an hrmult schema that is a copy of the hr schema:

1. In SQL*Plus, connect to the mult1.example.com database as an administrative user. See Oracle Database Administrator's Guide for instructions about connecting to a database in SQL*Plus.
2. Create a directory object to hold the export dump file and export log file. The directory object can point to any accessible directory on the computer system. For example, the following statement creates a directory object named `dp_hrmult_dir` that points to the `/usr/tmp` directory:

```
CREATE DIRECTORY dp_hrmult_dir AS '/usr/tmp';
```

Substitute an appropriate directory on your computer system.

3. Determine the current system change number (SCN) of the source database:

```
SELECT DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER FROM DUAL;
```

The SCN value returned by this query is specified for the `FLASHBACK_SCN` Data Pump export parameter in Step 5. Because the hr schema includes foreign key constraints between tables, the `FLASHBACK_SCN` export parameter, or a similar export parameter, must be specified during export.

4. Exit SQL*Plus.

5. On a command line at the `mult1.example.com` database site, use Data Pump to export the hr schema at the `mult1.example.com` database. Ensure that you specify the SCN value returned in Step 3 for the `FLASHBACK_SCN` parameter:

```
expdp system SCHEMAS=hr DIRECTORY=dp_hrmult_dir DUMPFILE=hrmult_schema.dmp FLASHBACK_SCN=flashback_scn_value
```

6. On a command line at the `mult1.example.com` database site, use Data Pump to import the import dump file `hrmult_schema.dmp`:

```
impdp system SCHEMAS=hr DIRECTORY=dp_hrmult_dir DUMPFILE=hrmult_schema.dmp REMAP_SCHEMA=hr:hrmult
```

7. In SQL*Plus, connect to the `mult1.example.com` database as an administrative user.

8. Assign a password to the new hrmult user at the `mult1.example.com` database using the `ALTER USER` statement.

Remember the password that you assign to the hrmult user so that you can log in as the user in the future.

3.4 Create Queues and Database Links

This section illustrates how to create queues and database links for an Oracle Streams replication environment that includes three Oracle databases. The remaining parts of this example depend on the queues and database links that you configure in this section.

Complete the following steps to create the queues and database links at all of the databases.

1. Show Output and Spool Results
2. Create the ANYDATA Queue at `mult1.example.com`
3. Create the Database Links at `mult1.example.com`
4. Prepare the Tables at `mult1.example.com` for Latest Time Conflict Resolution
5. Create the ANYDATA Queue at `mult2.example.com`
6. Create the Database Links at `mult2.example.com`
7. Create the ANYDATA Queue at `mult3.example.com`
8. **Create the Database Links at mult3.example.com**

9. **Check the Spool Results**

---

**Note:**

If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line after this note to the next "END OF SCRIPT" line into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to all of the databases in the environment.

/********************************************************** BEGINNING OF SCRIPT **********************************************************/

**Show Output and Spool Results**

Run `SET ECHO ON` and specify the spool file for the script. Check the spool file for errors after you run this script.

`*/`

`SET ECHO ON`

`SPOOL streams_setup_mult.out`

`*/`

**Create the ANYDATA Queue at mult1.example.com**

Connect as the Oracle Streams administrator at `mult1.example.com`.

`*/`

`CONNECT strmadmin@mult1.example.com`

`*/`

**Run the SET_UP_QUEUE procedure to create the following queues:**

- The `captured_mult1` queue to hold changes captured at the `mult1.example.com` database and propagated to other databases.
- The `from_mult2` queue to hold changes captured at the `mult2.example.com` database and propagated to the `mult1.example.com` database to be applied.
- The `from_mult3` queue to hold changes captured at the `mult3.example.com` database and propagated to the `mult1.example.com` database to be applied.

Running the `SET_UP_QUEUE` procedure performs the following actions for each queue:

- Creates a queue table that is owned by the Oracle Streams administrator (`strmadmin`) and that uses the default storage of this user.
- Creates an ANYDATA queue that is owned by the Oracle Streams administrator (`strmadmin`).
- Starts the queue.
BEGIN
  DBMSSTREAMSADM.SETUP_QUEUE(
    queue_table => ' strmadmin.captured_mult1_table',
    queue_name  => ' strmadmin.captured_mult1');
END;
/

BEGIN
  DBMSSTREAMSADM.SETUP_QUEUE(
    queue_table => ' strmadmin.from_mult2_table',
    queue_name  => ' strmadmin.from_mult2');
END;
/

BEGIN
  DBMSSTREAMSADM.SETUP_QUEUE(
    queue_table => ' strmadmin.from_mult3_table',
    queue_name  => ' strmadmin.from_mult3');
END;
/
/
/*

Create the Database Links at mult1.example.com
Create database links from the current database to the other databases in the environment.
*/

ACCEPT password PROMPT 'Enter password for user: ' HIDE

CREATE DATABASE LINK mult2.example.com CONNECT TO strmadmin
  IDENTIFIED BY &password USING 'mult2.example.com';

CREATE DATABASE LINK mult3.example.com CONNECT TO strmadmin
  IDENTIFIED BY &password USING 'mult3.example.com';
/
/
Prepare the Tables at mult1.example.com for Latest Time Conflict Resolution
This example will configure the tables in the hrmult schema for conflict resolution
based on the latest time for a transaction.

Connect to mult1.example.com as the hrmult user.
*/

CONNECT hrmult@mult1.example.com
/
/
Add a time column to each table in the hrmult schema.
*/

ALTER TABLE hrmult.countries ADD (time TIMESTAMP WITH TIME ZONE);
ALTER TABLE hrmult.departments ADD (time TIMESTAMP WITH TIME ZONE);
ALTER TABLE hrmult.employees ADD (time TIMESTAMP WITH TIME ZONE);
ALTER TABLE hrmult.job_history ADD (time TIMESTAMP WITH TIME ZONE);
ALTER TABLE hrmult.jobs ADD (time TIMESTAMP WITH TIME ZONE);
ALTER TABLE hrmult.locations ADD (time TIMESTAMP WITH TIME ZONE);
ALTER TABLE hrmult.regions ADD (time TIMESTAMP WITH TIME ZONE);

/

Create a trigger for each table in the hrmult schema to insert the time of a transaction for each row inserted or updated by the transaction.

CREATE OR REPLACE TRIGGER hrmult.insert_time_countries
BEFORE
    INSERT OR UPDATE ON hrmult.countries FOR EACH ROW
BEGIN
    IF :OLD.TIME IS NULL OR :OLD.TIME < SYSTIMESTAMP THEN
        :NEW.TIME := SYSTIMESTAMP;
    ELSE
        :NEW.TIME := :OLD.TIME + 1 / 86400;
    END IF;
END;
/

CREATE OR REPLACE TRIGGER hrmult.insert_time_departments
BEFORE
    INSERT OR UPDATE ON hrmult.departments FOR EACH ROW
BEGIN
    IF :OLD.TIME IS NULL OR :OLD.TIME < SYSTIMESTAMP THEN
        :NEW.TIME := SYSTIMESTAMP;
    ELSE
        :NEW.TIME := :OLD.TIME + 1 / 86400;
    END IF;
END;
/

CREATE OR REPLACE TRIGGER hrmult.insert_time_employees
BEFORE
    INSERT OR UPDATE ON hrmult.employees FOR EACH ROW
BEGIN
    IF :OLD.TIME IS NULL OR :OLD.TIME < SYSTIMESTAMP THEN
        :NEW.TIME := SYSTIMESTAMP;
    ELSE
        :NEW.TIME := :OLD.TIME + 1 / 86400;
    END IF;
END;
/

CREATE OR REPLACE TRIGGER hrmult.insert_time_job_history
BEFORE
    INSERT OR UPDATE ON hrmult.job_history FOR EACH ROW
BEGIN
    IF :OLD.TIME IS NULL OR :OLD.TIME < SYSTIMESTAMP THEN
        :NEW.TIME := SYSTIMESTAMP;
    ELSE
        :NEW.TIME := :OLD.TIME + 1 / 86400;
CREATE OR REPLACE TRIGGER hrmult.insert_time_jobs
BEFORE
INSERT OR UPDATE ON hrmult.jobs FOR EACH ROW
BEGIN
    IF :OLD.TIME IS NULL OR :OLD.TIME < SYSTIMESTAMP THEN
        :NEW.TIME := SYSTIMESTAMP;
    ELSE
        :NEW.TIME := :OLD.TIME + 1 / 86400;
    END IF;
END;
/

CREATE OR REPLACE TRIGGER hrmult.insert_time_locations
BEFORE
INSERT OR UPDATE ON hrmult.locations FOR EACH ROW
BEGIN
    IF :OLD.TIME IS NULL OR :OLD.TIME < SYSTIMESTAMP THEN
        :NEW.TIME := SYSTIMESTAMP;
    ELSE
        :NEW.TIME := :OLD.TIME + 1 / 86400;
    END IF;
END;
/

CREATE OR REPLACE TRIGGER hrmult.insert_time_regions
BEFORE
INSERT OR UPDATE ON hrmult.regions FOR EACH ROW
BEGIN
    IF :OLD.TIME IS NULL OR :OLD.TIME < SYSTIMESTAMP THEN
        :NEW.TIME := SYSTIMESTAMP;
    ELSE
        :NEW.TIME := :OLD.TIME + 1 / 86400;
    END IF;
END;
/

/*

Create the ANYDATA Queue at mult2.example.com
Connect as the Oracle Streams administrator at mult2.example.com.
*/

CONNECT strmadmin@mult2.example.com

/*

Run the SET_UP_QUEUE procedure to create the following queues:

- The captured_mult2 queue to hold changes captured at the mult2.example.com database and propagated to other databases.
- The from_mult1 queue to hold changes captured at the mult1.example.com database and propagated to the mult2.example.com database to be applied.
*/
The `from_mult3` queue to hold changes captured at the `mult3.example.com` database and propagated to the `mult2.example.com` database to be applied.

Running the `SET_UP_QUEUE` procedure performs the following actions for each queue:

- Creates a queue table that is owned by the Oracle Streams administrator (`strmadmin`) and that uses the default storage of this user.
- Creates an `ANYDATA` queue that is owned by the Oracle Streams administrator (`strmadmin`).
- Starts the queue.

```
BEGIN
  DBMS_STREAMS_ADM.SET_UP_QUEUE(
    queue_table  => 'strmadmin.captured_mult2_table',
    queue_name   => 'strmadmin.captured_mult2');
END;
/
BEGIN
  DBMS_STREAMS_ADM.SET_UP_QUEUE(
    queue_table  => 'strmadmin.from_mult1_table',
    queue_name   => 'strmadmin.from_mult1');
END;
/
BEGIN
  DBMS_STREAMS_ADM.SET_UP_QUEUE(
    queue_table  => 'strmadmin.from_mult3_table',
    queue_name   => 'strmadmin.from_mult3');
END;
/
/*

Create the Database Links at `mult2.example.com`
Create database links from the current database to the other databases in the environment.

CREATE DATABASE LINK mult1.example.com CONNECT TO strmadmin
  IDENTIFIED BY &password USING 'mult1.example.com';

CREATE DATABASE LINK mult3.example.com CONNECT TO strmadmin
  IDENTIFIED BY &password USING 'mult3.example.com';

/*

Create the ANYDATA Queue at `mult3.example.com`
Connect as the Oracle Streams administrator at `mult3.example.com`.

CONNECT strmadmin@mult3.example.com
/*
Run the `SET_UP_QUEUE` procedure to create the following queues:

- The `captured_mult3` queue to hold changes captured at the `mult3.example.com` database and propagated to other databases.
- The `from_mult1` queue to hold changes captured at the `mult1.example.com` database and propagated to the `mult3.example.com` database to be applied.
- The `from_mult2` queue to hold changes captured at the `mult2.example.com` database and propagated to the `mult3.example.com` database to be applied.

Running the `SET_UP_QUEUE` procedure performs the following actions for each queue:

- Creates a queue table that is owned by the Oracle Streams administrator (`strmadmin`) and that uses the default storage of this user.
- Creates an `ANYDATA` queue that is owned by the Oracle Streams administrator (`strmadmin`).
- Starts the queue.

```sql
BEGIN
  DBMS_STREAMS_ADM.SET_UP_QUEUE(
    queue_table  => 'strmadmin.captured_mult3_table',
    queue_name   => 'strmadmin.captured_mult3');
END;
/

BEGIN
  DBMS_STREAMS_ADM.SET_UP_QUEUE(
    queue_table  => 'strmadmin.from_mult1_table',
    queue_name   => 'strmadmin.from_mult1');
END;
/

BEGIN
  DBMS_STREAMS_ADM.SET_UP_QUEUE(
    queue_table  => 'strmadmin.from_mult2_table',
    queue_name   => 'strmadmin.from_mult2');
END;
/

CREATE DATABASE LINK mult1.example.com
CONNECT TO strmadmin
IDENTIFIED BY &password USING 'mult1.example.com';

CREATE DATABASE LINK mult2.example.com
CONNECT TO strmadmin
IDENTIFIED BY &password USING 'mult2.example.com';
```
Check the Spool Results
Check the streams_setup_mult.out spool file to ensure that all actions finished successfully after this script is completed.

*/

SET ECHO OFF
SPOOL OFF

/**************************** END OF SCRIPT ****************************/

3.5 Example Script for Configuring N-Way Replication

Complete the following steps to configure an Oracle Streams n-way replication environment.

1. Show Output and Spool Results
2. Specify Supplemental Logging at mult1.example.com
3. Create the Capture Process at mult1.example.com
4. Create One Apply Process at mult1.example.com for Each Source Database
5. Configure Latest Time Conflict Resolution at mult1.example.com
6. Configure Propagation at mult1.example.com
7. Create the Capture Process at mult2.example.com.
8. Set the Instantiation SCN for mult2.example.com at the Other Databases
9. Create One Apply Process at mult2.example.com for Each Source Database
10. Configure Propagation at mult2.example.com
11. Create the Capture Process at mult3.example.com
12. Set the Instantiation SCN for mult3.example.com at the Other Databases
13. Create One Apply Process at mult3.example.com for Each Source Database
14. Configure Propagation at mult3.example.com
15. Instantiate the hrmult Schema at mult2.example.com
16. Instantiate the hrmult Schema at mult3.example.com
17. Configure Latest Time Conflict Resolution at mult2.example.com
18. Start the Apply Processes at mult2.example.com
19. Configure Latest Time Conflict Resolution at mult3.example.com
20. Start the Apply Processes at mult3.example.com
21. Start the Apply Processes at mult1.example.com
22. Start the Capture Process at mult1.example.com
23. Start the Capture Process at mult2.example.com
24. Start the Capture Process at mult3.example.com
25. Check the Spool Results
Show Output and Spool Results
Run SET ECHO ON and specify the spool file for the script. Check the spool file for errors after you run this script.

```
SET ECHO ON
SPOOL streams_mult.out
```

Specify Supplemental Logging at mult1.example.com
Connect to mult1.example.com as the strmadmin user.

```
CONNECT strmadmin@mult1.example.com
```

Specify an unconditional supplemental log group that includes the primary key for each table and the column list for each table, as specified in "Configure Latest Time Conflict Resolution at mult1.example.com". Because the column list for each table includes all of the columns of each table except for its primary key, this step creates a supplemental log group for each table that includes all of the columns in the table.

Note:
- For convenience, this example includes the primary key column(s) for each table and the columns used for update conflict resolution in a single unconditional log group. You can choose to place the primary key column(s) for each table in an unconditional log group and the columns used for update conflict resolution in a conditional log group.
- You do not need to specify supplemental logging explicitly at mult2.example.com and mult3.example.com in this example. When you use Data Pump to instantiate the tables in the hrmult schema at these databases later in this example, the supplemental logging specifications at mult1.example.com are retained at mult2.example.com and mult3.example.com.
See Also:

Oracle Streams Replication Administrator’s Guide

/ *
ALTER TABLE hrmult.countries ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS;
ALTER TABLE hrmult.departments ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS;
ALTER TABLE hrmult.employees ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS;
ALTER TABLE hrmult.jobs ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS;
ALTER TABLE hrmult.job_history ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS;
ALTER TABLE hrmult.locations ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS;
ALTER TABLE hrmult.regions ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS;
/

Create the Capture Process at mult1.example.com
Create the capture process to capture changes to the entire hrmult schema at
mult1.example.com. This step also prepares the hrmult schema at mult1.example.com for
instantiation. After this step is complete, users can modify tables in the hrmult schema
at mult1.example.com.

/ *
BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
    schema_name    => 'hrmult',
    streams_type   => 'capture',
    streams_name   => 'capture_hrmult',
    queue_name     => 'strmadmin.captured_mult1',
    include_dml    => TRUE,
    include_ddl    => TRUE,
    inclusion_rule => TRUE);
END;
/

Create One Apply Process at mult1.example.com for Each Source Database
Configure mult1.example.com to apply changes to the hrmult schema at
mult2.example.com.

/ *
BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
    schema_name    => 'hrmult',
    streams_type   => 'apply',
    streams_name   => 'apply_from_mult2',
    queue_name     => 'strmadmin.from_mult2',
    include_dml    => TRUE,
include_ddl => TRUE,
source_database => 'mult2.example.com',
inclusion_rule => TRUE);
END;
/
/

Configure mult1.example.com to apply changes to the hrmult schema at mult3.example.com.
*/

BEGIN
DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
    schema_name     => 'hrmult',
    streams_type    => 'apply',
    streams_name    => 'apply_from_mult3',
    queue_name      => 'strmadmin.from_mult3',
    include_dml     => TRUE,
    include_ddl     => TRUE,
    source_database => 'mult3.example.com',
    inclusion_rule  => TRUE);
END;
/
/

Configure Latest Time Conflict Resolution at mult1.example.com
Specify an update conflict handler for each table in the hrmult schema. For each table, designate the time column as the resolution column for a MAXIMUM conflict handler. When an update conflict occurs, such an update conflict handler applies the transaction with the latest (or greater) time and discards the transaction with the earlier (or lesser) time. The column lists include all columns for each table, except for the primary key, because this example assumes that primary key values are never updated.
*/

DECLARE
    cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
    cols(1) := 'country_name';
    cols(2) := 'region_id';
    cols(3) := 'time';
    DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
        object_name           => 'hrmult.countries',
        method_name           => 'MAXIMUM',
        resolution_column     => 'time',
        column_list           => cols);
END;
/

DECLARE
    cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
    cols(1) := 'department_name';
    cols(2) := 'manager_id';
    cols(3) := 'location_id';
DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
    object_name => 'hrmult.departments',
    method_name => 'MAXIMUM',
    resolution_column => 'time',
    column_list => cols);
END;
/

DECLARE
    cols DBMS_UTILITY.NAME_ARRAY;
BEGIN
    cols(1) := 'first_name';
    cols(2) := 'last_name';
    cols(3) := 'email';
    cols(4) := 'phone_number';
    cols(5) := 'hire_date';
    cols(6) := 'job_id';
    cols(7) := 'salary';
    cols(8) := 'commission_pct';
    cols(9) := 'manager_id';
    cols(10) := 'department_id';
    cols(11) := 'time';
DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
    object_name => 'hrmult.employees',
    method_name => 'MAXIMUM',
    resolution_column => 'time',
    column_list => cols);
END;
/

DECLARE
    cols DBMS_UTILITY.NAME_ARRAY;
BEGIN
    cols(1) := 'job_title';
    cols(2) := 'min_salary';
    cols(3) := 'max_salary';
    cols(4) := 'time';
DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
    object_name => 'hrmult.jobs',
    method_name => 'MAXIMUM',
    resolution_column => 'time',
    column_list => cols);
END;
/

DECLARE
    cols DBMS_UTILITY.NAME_ARRAY;
BEGIN
    cols(1) := 'employee_id';
    cols(2) := 'start_date';
    cols(3) := 'end_date';
    cols(4) := 'job_id';
    cols(5) := 'department_id';
    cols(6) := 'time';
DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
    object_name => 'hrmult.job_history',
    method_name => 'MAXIMUM',
    resolution_column => 'time',
    column_list => cols);
END;
column_list => cols);
END;
/

DECLARE
  cols DBMS_UTILITY.NAME_ARRAY;
BEGIN
  cols(1) := 'street_address';
  cols(2) := 'postal_code';
  cols(3) := 'city';
  cols(4) := 'state_province';
  cols(5) := 'country_id';
  cols(6) := 'time';
  DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
    object_name => 'hrmult.locations',
    method_name => 'MAXIMUM',
    resolution_column => 'time',
    column_list => cols);
END;
/

DECLARE
  cols DBMS_UTILITY.NAME_ARRAY;
BEGIN
  cols(1) := 'region_name';
  cols(2) := 'time';
  DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
    object_name => 'hrmult.regions',
    method_name => 'MAXIMUM',
    resolution_column => 'time',
    column_list => cols);
END;
/
/

Configure Propagation at mult1.example.com
Configure and schedule propagation of DML and DDL changes in the hrmult schema from the queue at mult1.example.com to the queue at mult2.example.com.
*/

BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES(
    schema_name => 'hrmult',
    streams_name => 'mult1_to_mult2',
    source_queue_name => 'strmadmin.captured_mult1',
    destination_queue_name => 'strmadmin.from_mult1@mult2.example.com',
    include_dml => TRUE,
    include_ddl => TRUE,
    source_database => 'mult1.example.com',
    inclusion_rule => TRUE,
    queue_to_queue => TRUE);
END;
/
/

Configure and schedule propagation of DML and DDL changes in the hrmult schema from the queue at mult1.example.com to the queue at mult3.example.com.
BEGIN
DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES(
    schema_name => 'hrmult',
    streams_name => 'mult1_to_mult3',
    source_queue_name => 'strmadmin.captured_mult1',
    destination_queue_name => 'strmadmin.from_mult1@mult3.example.com',
    include_dml => TRUE,
    include_ddl => TRUE,
    source_database => 'mult1.example.com',
    inclusion_rule => TRUE,
    queue_to_queue => TRUE);
END;
/
/*
Create the Capture Process at mult2.example.com.
Connect to mult2.example.com as the strmadmin user.
*/
CONNECT strmadmin@mult2.example.com
/
/*
Create the capture process to capture changes to the entire hrmult schema at
mult2.example.com. This step also prepares the hrmult schema at mult2.example.com for
instantiation.
*/
BEGIN
DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
    schema_name => 'hrmult',
    streams_type => 'capture',
    streams_name => 'capture_hrmult',
    queue_name => 'strmadmin.captured_mult2',
    include_dml => TRUE,
    include_ddl => TRUE,
    inclusion_rule => TRUE);
END;
/
/*
Set the Instantiation SCN for mult2.example.com at the Other Databases
In this example, the hrmult schema already exists at all of the databases. The tables
in the schema exist only at mult1.example.com until they are instantiated at
mult2.example.com and mult3.example.com in Step Instantiate the hrmult Schema at
mult3.example.com. The instantiation is done using an import of the tables from
mult1.example.com. These import operations set the schema instantiation SCNs for
mult1.example.com at mult2.example.com and mult3.example.com automatically.
However, the instantiation SCNs for mult2.example.com and mult3.example.com are not
set automatically at the other sites in the environment. This step sets the schema
instantiation SCN for mult2.example.com manually at mult1.example.com and
mult3.example.com. The current SCN at mult2.example.com is obtained by using the
GET_SYSTEM_CHANGE_NUMBER function in the DBMS_FLASHBACK package at mult2.example.com.
*/
This SCN is used at mult1.example.com and mult3.example.com to run the SET_SCHEMA_INSTANTIATION_SCN procedure in the DBMS_APPLY_ADM package. The SET_SCHEMA_INSTANTIATION_SCN procedure controls which DDL LCRs for a schema are ignored by an apply process and which DDL LCRs for a schema are applied by an apply process. If the commit SCN of a DDL LCR for a database object in a schema from a source database is less than or equal to the instantiation SCN for that database object at some destination database, then the apply process at the destination database disregards the DDL LCR. Otherwise, the apply process applies the DDL LCR. Because you are running the SET_SCHEMA_INSTANTIATION_SCN procedure before the tables are instantiated at mult2.example.com, and because the local capture process is configured already, you do not need to run the SET_TABLE_INSTANTIATION_SCN for each table after the instantiation. In this example, an apply process at both mult1.example.com and mult3.example.com will apply transactions to the tables in the hrmult schema with SCNs that were committed after the SCN obtained in this step.

Note:

- In a case where you are instantiating a schema that does not exist, you can set the global instantiation SCN instead of the schema instantiation SCN.
- In a case where the tables are instantiated before you set the instantiation SCN, you must set the schema instantiation SCN and the instantiation SCN for each table in the schema.

/*
DECLARE
  iscn NUMBER; -- Variable to hold instantiation SCN value
BEGIN
  iscn := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER();
  DBMS_APPLY_ADM.SET_SCHEMA_INSTANTIATION_SCN@MULT1.EXAMPLE.COM{
    source_schema_name => 'hrmult',
    source_database_name => 'mult2.example.com',
    instantiation_scn => iscn};
  DBMS_APPLY_ADM.SET_SCHEMA_INSTANTIATION_SCN@MULT3.EXAMPLE.COM{
    source_schema_name => 'hrmult',
    source_database_name => 'mult2.example.com',
    instantiation_scn => iscn};
END;
/
*/

Create One Apply Process at mult2.example.com for Each Source Database

Configure mult2.example.com to apply changes to the hrmult schema at mult1.example.com.

/*
BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_RULES{

*/
Configure mult2.example.com to apply changes to the hrmult schema at mult3.example.com.

Configure Propagation at mult2.example.com
Configure and schedule propagation of DML and DDL changes in the hrmult schema from the queue at mult2.example.com to the queue at mult1.example.com.

Configure and schedule propagation of DML and DDL changes in the hrmult schema from the queue at mult2.example.com to the queue at mult3.example.com.
BEGIN

DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES(
    schema_name => 'hrmult',
    streams_name => 'mult2_to_mult3',
    source_queue_name => 'strmadmin.captured_mult2',
    destination_queue_name => 'strmadmin.from_mult2@mult3.example.com',
    include_dml => TRUE,
    include_ddl => TRUE,
    source_database => 'mult2.example.com',
    inclusion_rule => TRUE,
    queue_to_queue => TRUE);

END;
/

CREATE the Capture Process at mult3.example.com
Connect to mult3.example.com as the strmadmin user.

BEGIN

DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
    schema_name => 'hrmult',
    streams_type => 'capture',
    streams_name => 'capture_hrmult',
    queue_name => 'strmadmin.captured_mult3',
    include_dml => TRUE,
    include_ddl => TRUE,
    inclusion_rule => TRUE);

END;
/

/*
Set the Instantiation SCN for mult3.example.com at the Other Databases
In this example, the hrmult schema already exists at all of the databases. The tables in the schema exist only at mult1.example.com until they are instantiated at mult2.example.com and mult3.example.com. The instantiation is done using an import of the tables from mult1.example.com. These import operations set the schema instantiation SCNs for mult1.example.com at mult2.example.com and mult3.example.com automatically. However, the instantiation SCNs for mult2.example.com and mult3.example.com are not set automatically at the other sites in the environment. This step sets the schema instantiation SCN for mult3.example.com manually at mult1.example.com and mult2.example.com. The current SCN at mult3.example.com is obtained by using the GET_SYSTEM_CHANGE_NUMBER function in the DBMS_FLASHBACK package at mult3.example.com.
*/
This SCN is used at mult1.example.com and mult2.example.com to run the `SET_SCHEMA_INSTANTIATION_SCN` procedure in the `DBMS_APPLY_ADM` package. The `SET_SCHEMA_INSTANTIATION_SCN` procedure controls which DDL LCRs for a schema are ignored by an apply process and which DDL LCRs for a schema are applied by an apply process. If the commit SCN of a DDL LCR for a database object in a schema from a source database is less than or equal to the instantiation SCN for that database object at some destination database, then the apply process at the destination database disregards the DDL LCR. Otherwise, the apply process applies the DDL LCR.

Because you are running the `SET_SCHEMA_INSTANTIATION_SCN` procedure before the tables are instantiated at mult3.example.com, and because the local capture process is configured already, you do not need to run the `SET_TABLE_INSTANTIATION_SCN` for each table after the instantiation. In this example, an apply process at both mult1.example.com and mult2.example.com will apply transactions to the tables in the hrml mult  schema with SCNs that were committed after the SCN obtained in this step.

> **Note:**

- In a case where you are instantiating a schema that does not exist, you can set the global instantiation SCN instead of the schema instantiation SCN.
- In a case where the tables are instantiated before you set the instantiation SCN, you must set the schema instantiation SCN and the instantiation SCN for each table in the schema.

```sql
DECLARE
  iscn NUMBER;         -- Variable to hold instantiation SCN value
BEGIN
  iscn := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER();
  DBMS_APPLY_ADM.SET_SCHEMA_INSTANTIATION_SCN@MULT1.EXAMPLE.COM(
    source_schema_name    => 'hrml mult',
    source_database_name  => 'mult3.example.com',
    instantiation_scn     => iscn);
  DBMS_APPLY_ADM.SET_SCHEMA_INSTANTIATION_SCN@MULT2.EXAMPLE.COM(
    source_schema_name    => 'hrml mult',
    source_database_name  => 'mult3.example.com',
    instantiation_scn     => iscn);
END;
/

CREATE ONE APPLY PROCESS AT mult3.example.com FOR EACH SOURCE DATABASE
Configure mult3.example.com to apply changes to the hrml mult schema at mult1.example.com.

```
Configure mult3.example.com to apply changes to the hrmult schema at mult2.example.com.

Configure Propagation at mult3.example.com
Configure and schedule propagation of DML and DDL changes in the hrmult schema from the queue at mult3.example.com to the queue at mult1.example.com.
BEGIN
DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES(
    schema_name             => 'hrmult',
    streams_name            => 'mult3_to_mult2',
    source_queue_name       => 'strmadmin.captured_mult3',
    destination_queue_name  => 'strmadmin.from_mult3@mult2.example.com',
    include_dml             => TRUE,
    include_ddl             => TRUE,
    source_database         => 'mult3.example.com',
    inclusion_rule          => TRUE,
    queue_to_queue          => TRUE);
END;
/

Instantiation the hrmult Schema at mult2.example.com
This example performs a network Data Pump import of the hrmult schema from
mult1.example.com to mult2.example.com. A network import means that Data Pump
imports the database objects in the schema from mult1.example.com without using an
export dump file.

See Also:
Oracle Database Utilities for information about performing an import

Connect to mult2.example.com as the strmadmin user.
/
CONNECT strmadmin@mult2.example.com
/

This example will do a schema-level import using the DBMS_DATAPUMP package. For
simplicity, exceptions from any of the API calls will not be trapped. However, Oracle
recommends that you define exception handlers and call GET_STATUS to retrieve more
detailed error information if a failure occurs. If you want to monitor the import, then
query the DBA_DATAPUMP_JOBS data dictionary view at the import database.
/

SET SERVEROUTPUT ON
DECLARE
    h1                NUMBER;         -- Data Pump job handle
    mult2_instantscn NUMBER;         -- Variable to hold current source SCN
    job_state         VARCHAR2(30);   -- To keep track of job state
    js                ku$_JobStatus;  -- The job status from GET_STATUS
    sts               ku$_Status;     -- The status object returned by GET_STATUS
    job_not_exist    exception;
    pragma exception_init(job_not_exist, -31626);
BEGIN
    -- Create a (user-named) Data Pump job to do a schema-level import.
    h1 := DBMS_DATAPUMP.OPEN(
Instantiate the hrmult Schema at mult3.example.com

This example performs a network Data Pump import of the hrmult schema from mult1.example.com to mult3.example.com. A network import means that Data Pump imports the database objects in the schema from mult1.example.com without using an export dump file.

---

See Also:

Oracle Database Utilities for information about performing an import

Connect to mult3.example.com as the strmadmin user.
CONNECT strmadmin@mult3.example.com

This example will do a table import using the DBMS_DATAPOU package. For simplicity, exceptions from any of the API calls will not be trapped. However, Oracle recommends that you define exception handlers and call GET_STATUS to retrieve more detailed error information if a failure occurs. If you want to monitor the import, then query the DBA_DATAPUMP_JOBS data dictionary view at the import database.

SET SERVEROUTPUT ON
DECLARE
  h1 NUMBER; -- Data Pump job handle
  mult3_instantscn NUMBER; -- Variable to hold current source SCN
  job_state VARCHAR2(30); -- To keep track of job state
  js ku$_JobStatus; -- The job status from GET_STATUS
  sts ku$_Status; -- The status object returned by GET_STATUS
  job_not_exist exception;
  pragma exception_init(job_not_exist, -31626);
BEGIN
  -- Create a (user-named) Data Pump job to do a schema-level import.
  h1 := DBMS_DATAPUMP.OPEN(
    operation => 'IMPORT',
    job_mode => 'SCHEMA',
    remote_link => 'MULT1.EXAMPLE.COM',
    job_name => 'dp_mult3');
  -- A metadata filter is used to specify the schema that owns the tables
  -- that will be imported.
  DBMS_DATAPUMP.METADATA_FILTER(
    handle => h1,
    name => 'SCHEMA_EXPR',
    value => '=HRMULT');
  -- Get the current SCN of the source database, and set the FLASHBACK_SCN
  -- parameter to this value to ensure consistency between all of the
  -- objects in the schema.
  mult3_instantscn := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER@mult1.example.com();
  DBMS_DATAPUMP.SET_PARAMETER(
    handle => h1,
    name => 'FLASHBACK_SCN',
    value => mult3_instantscn);
  -- Start the job.
  DBMS_DATAPUMP.START_JOB(h1);
  -- The import job should be running. In the following loop, the job
  -- is monitored until it completes.
  job_state := 'UNDEFINED';
  BEGIN
    WHILE (job_state != 'COMPLETED') AND (job_state != 'STOPPED') LOOP
      sts:=DBMS_DATAPUMP.GET_STATUS(
        handle => h1,
        mask => DBMS_DATAPUMP.KU$_STATUS_JOB_ERROR +
                DBMS_DATAPUMP.KU$_STATUS_JOB_STATUS +
                DBMS_DATAPUMP.KU$_STATUS_WIP,
        timeout => -1);
      js := sts.job_status;
      DBMS_LOCK.SLEEP(10);
job_state := js.state;
END LOOP;
-- Gets an exception when job no longer exists
EXCEPTION WHEN job_not_exist THEN
    DBMS_OUTPUT.PUT_LINE('Data Pump job has completed');
    DBMS_OUTPUT.PUT_LINE('Instantiation SCN: ' || mult3_instantscn);
END;
END;
/

/*
Configure Latest Time Conflict Resolution at mult2.example.com
Connect to mult2.example.com as the strmadmin user.
*/
CONNECT strmadmin@mult2.example.com
/

Specify an update conflict handler for each table in the hrmult schema. For each
table, designate the time column as the resolution column for a MAXIMUM conflict
handler. When an update conflict occurs, such an update conflict handler applies the
transaction with the latest (or greater) time and discards the transaction with the earlier (or lesser) time.
*/

DECLARE
    cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
    cols(1) := 'country_name';
    cols(2) := 'region_id';
    cols(3) := 'time';
    DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(       
        object_name           =>  'hrmult.countries',
        method_name           =>  'MAXIMUM',
        resolution_column     =>  'time',
        column_list           =>  cols);
END;
/

DECLARE
    cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
    cols(1) := 'department_name';
    cols(2) := 'manager_id';
    cols(3) := 'location_id';
    cols(4) := 'time';
    DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(       
        object_name           =>  'hrmult.departments',
        method_name           =>  'MAXIMUM',
        resolution_column     =>  'time',
        column_list           =>  cols);
END;
/

DECLARE
cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
  cols(1) := 'first_name';
  cols(2) := 'last_name';
  cols(3) := 'email';
  cols(4) := 'phone_number';
  cols(5) := 'hire_date';
  cols(6) := 'job_id';
  cols(7) := 'salary';
  cols(8) := 'commission_pct';
  cols(9) := 'manager_id';
  cols(10) := 'department_id';
  cols(11) := 'time';
  DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
    object_name => 'hrmult.employees',
    method_name => 'MAXIMUM',
    resolution_column => 'time',
    column_list => cols);
END;
/

DECLARE
  cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
  cols(1) := 'job_title';
  cols(2) := 'min_salary';
  cols(3) := 'max_salary';
  cols(4) := 'time';
  DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
    object_name => 'hrmult.jobs',
    method_name => 'MAXIMUM',
    resolution_column => 'time',
    column_list => cols);
END;
/

DECLARE
  cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
  cols(1) := 'employee_id';
  cols(2) := 'start_date';
  cols(3) := 'end_date';
  cols(4) := 'job_id';
  cols(5) := 'department_id';
  cols(6) := 'time';
  DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
    object_name => 'hrmult.job_history',
    method_name => 'MAXIMUM',
    resolution_column => 'time',
    column_list => cols);
END;
/

DECLARE
  cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
  cols(1) := 'street_address';
  cols(2) := 'postal_code';
  cols(3) := 'city';
 cols(4) := 'state_province';
cols(5) := 'country_id';
cols(6) := 'time';
DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
    object_name => 'hrmult.locations',
    method_name => 'MAXIMUM',
    resolution_column => 'time',
    column_list => cols);
END;
/

DECLARE
  cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
  cols(1) := 'region_name';
cols(2) := 'time';
DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
    object_name => 'hrmult.regions',
    method_name => 'MAXIMUM',
    resolution_column => 'time',
    column_list => cols);
END;
/
/

Start the Apply Processes at mult2.example.com
Start both of the apply processes at mult2.example.com.
*/
BEGIN
  DBMS_APPLY_ADM.START_APPLY(
    apply_name => 'apply_from_mult1');
END;
/

BEGIN
  DBMS_APPLY_ADM.START_APPLY(
    apply_name => 'apply_from_mult3');
END;
/
/
Configure Latest Time Conflict Resolution at mult3.example.com
Connect to mult3.example.com as the strmadmin user.
*/
CONNECT strmadmin@mult3.example.com
/
/
Specify an update conflict handler for each table in the hrmult schema. For each table, designate the time column as the resolution column for a MAXIMUM conflict handler. When an update conflict occurs, such an update conflict handler applies the transaction with the latest (or greater) time and discards the transaction with the earlier (or lesser) time.
DECLARE
    cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
    cols(1) := 'country_name';
    cols(2) := 'region_id';
    cols(3) := 'time';
    DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
        object_name       => 'hrmult.countries',
        method_name       => 'MAXIMUM',
        resolution_column => 'time',
        column_list       => cols);
END;
/

DECLARE
    cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
    cols(1) := 'department_name';
    cols(2) := 'manager_id';
    cols(3) := 'location_id';
    cols(4) := 'time';
    DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
        object_name       => 'hrmult.departments',
        method_name       => 'MAXIMUM',
        resolution_column => 'time',
        column_list       => cols);
END;
/

DECLARE
    cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
    cols(1) := 'first_name';
    cols(2) := 'last_name';
    cols(3) := 'email';
    cols(4) := 'phone_number';
    cols(5) := 'hire_date';
    cols(6) := 'job_id';
    cols(7) := 'salary';
    cols(8) := 'commission_pct';
    cols(9) := 'manager_id';
    cols(10) := 'department_id';
    cols(11) := 'time';
    DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
        object_name       => 'hrmult.employees',
        method_name       => 'MAXIMUM',
        resolution_column => 'time',
        column_list       => cols);
END;
/

DECLARE
    cols  DBMS_UTILITY.NAME_ARRAY;
BEGIN
    cols(1) := 'job_title';
    cols(2) := 'min_salary';
    cols(3) := 'max_salary';
    DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
        object_name       => 'hrmult.jobs',
        method_name       => 'MAXIMUM',
        resolution_column => 'time',
        column_list       => cols);
END;
cols(4) := 'time';
DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
  object_name => 'hrmult.jobs,'
  method_name => 'MAXIMUM',
  resolution_column => 'time',
  column_list => cols);
END;
/

DECLARE
  cols DBMS_UTILITY.NAME_ARRAY;
BEGIN
  cols(1) := 'employee_id';
  cols(2) := 'start_date';
  cols(3) := 'end_date';
  cols(4) := 'job_id';
  cols(5) := 'department_id';
  cols(6) := 'time';
DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
  object_name => 'hrmult.job_history',
  method_name => 'MAXIMUM',
  resolution_column => 'time',
  column_list => cols);
END;
/

DECLARE
  cols DBMS_UTILITY.NAME_ARRAY;
BEGIN
  cols(1) := 'street_address';
  cols(2) := 'postal_code';
  cols(3) := 'city';
  cols(4) := 'state_province';
  cols(5) := 'country_id';
  cols(6) := 'time';
DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
  object_name => 'hrmult.locations',
  method_name => 'MAXIMUM',
  resolution_column => 'time',
  column_list => cols);
END;
/

DECLARE
  cols DBMS_UTILITY.NAME_ARRAY;
BEGIN
  cols(1) := 'region_name';
  cols(2) := 'time';
DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
  object_name => 'hrmult.regions',
  method_name => 'MAXIMUM',
  resolution_column => 'time',
  column_list => cols);
END;
/
/

Start the Apply Processes at mult3.example.com
Start both of the apply processes at mult3.example.com.
BEGIN
  DBMS_APPLY_ADM.START_APPLY(
    apply_name => 'apply_from_mult1');
END;
/
BEGIN
  DBMS_APPLY_ADM.START_APPLY(
    apply_name => 'apply_from_mult2');
END;
/
BEGIN
  DBMS_APPLY_ADM.START_APPLY(
    apply_name => 'apply_from_mult3');
END;
/

Start the Apply Processes at mult1.example.com
Connect to mult1.example.com as the strmadmin user.

CONNECT strmadmin@mult1.example.com
/

Start both of the apply processes at mult1.example.com.

/
BEGIN
  DBMS_APPLY_ADM.START_APPLY(
    apply_name => 'apply_from_mult2');
END;
/
BEGIN
  DBMS_APPLY_ADM.START_APPLY(
    apply_name => 'apply_from_mult3');
END;
/

Start the Capture Process at mult1.example.com
Start the capture process at mult1.example.com.

/
BEGIN
  DBMS_CAPTURE_ADM.START_CAPTURE(
    capture_name => 'capture_hrmult');
END;
/
/
Start the Capture Process at mult2.example.com
Connect to mult2.example.com as the strmadmin user.
CONNECT strmadmin@mult2.example.com
/

Start the capture process at mult2.example.com.
*/
BEGIN
    DBMS_CAPTURE_ADM.START_CAPTURE(
        capture_name => 'capture_hrmult');
END;
/
/
Start the Capture Process at mult3.example.com
Connect to mult3.example.com as the strmadmin user.
*/
CONNECT strmadmin@mult3.example.com
/

Start the capture process at mult3.example.com.
*/
BEGIN
    DBMS_CAPTURE_ADM.START_CAPTURE(
        capture_name => 'capture_hrmult');
END;
/

SET ECHO OFF
/

Check the Spool Results
Check the streams_mult.out spool file to ensure that all actions finished successfully after this script is completed.
*/
SET ECHO OFF
SPOOL OFF

/**************************** END OF SCRIPT ******************************/

See Also:

Oracle Streams Replication Administrator's Guide for general instructions that explain how to add database objects or databases to the replication environment
3.6 Make DML and DDL Changes to Tables in the hrmult Schema

You can make DML and DDL changes to the tables in the hrmult schema at any of the databases in the environment. These changes will be replicated to the other databases in the environment, and you can run queries to view the replicated data.

For example, complete the following steps to make DML changes to the hrmult.employees table at mult1.example.com and mult2.example.com. To see the update conflict handler you configured earlier resolve an update conflict, you can make a change to the same row in these two databases and commit the changes at nearly the same time. You can query the changed row at each database in the environment to confirm that the changes were captured, propagated, and applied correctly.

You can also make a DDL change to the hrmult.jobs table at mult3.example.com and then confirm that the change was captured at mult3.example.com, propagated to the other databases in the environment, and applied at these databases.

Make a DML Change to hrmult.employees at mult.example.com and mult2.example.com

Make the following changes. To simulate a conflict, try to commit them at nearly the same time, but commit the change at mult2.example.com after you commit the change at mult1.example.com. The update conflict handler at each database will resolve the conflict.

CONNECT hrmult@mult1.example.com
Enter password: password

UPDATE hrmult.employees SET salary=9000 WHERE employee_id=206;
COMMIT;

CONNECT hrmult@mult2.example.com
Enter password: password

UPDATE hrmult.employees SET salary=10000 WHERE employee_id=206;
COMMIT;

Alter the hrmult.jobs Table at mult3.example.com

Alter the hrmult.jobs table by renaming the job_title column to job_name:

CONNECT hrmult@mult3.example.com
Enter password: password

ALTER TABLE hrmult.jobs RENAME COLUMN job_title TO job_name;

Query the hrmult.employees Table at Each Database

After some time passes to allow for capture, propagation, and apply of the changes performed in Step Make a DML Change to hrmult.employees at mult.example.com and mult2.example.com, run the following query to confirm that the UPDATE changes have been applied at each database.

CONNECT hrmult@mult1.example.com
Enter password: password

SELECT salary FROM hrmult.employees WHERE employee_id=206;
All of the queries should show 10000 for the value of the salary. The update conflict handler at each database has resolved the conflict by using the latest change to the row. In this case, the latest change to the row was made at the mult2.example.com database in Step Make a DML Change to hrmult.employees at mult.example.com and mult2.example.com.

Describe the hrmult.jobs Table at Each Database
After some time passes to allow for capture, propagation, and apply of the change performed in Step Alter the hrmult.jobs Table at mult3.example.com, describe the hrmult.jobs table at each database to confirm that the ALTER TABLE change was propagated and applied correctly.

Each database should show job_name as the second column in the table.
4

Single-Database Capture and Apply Example

This chapter illustrates an example of a single database that captures changes to a table with a capture process, reenqueues the captured changes into a queue, and then uses a procedure DML handler during apply to insert a subset of the changes into a different table.

The following topics describe configuring an example single-database capture and apply example:

- Overview of the Single-Database Capture and Apply Example
- Prerequisites
- Set Up the Environment
- Configure Capture and Apply
- Make DML Changes, Query for Results, and Dequeue Messages

4.1 Overview of the Single-Database Capture and Apply Example

The example in this chapter illustrates using Oracle Streams to capture and apply data manipulation language (DML) changes at a single database named cpap.example.com. Specifically, this example captures DML changes to the employees table in the hr schema, placing row logical change records (LCRs) into a queue named streams_queue. Next, an apply process dequeues these row LCRs from the same queue, reenqueues them into this queue, and sends them to a procedure DML handler.

When the row LCRs are captured, they reside in the buffered queue and cannot be dequeued explicitly. After the row LCRs are reenqueued during apply, they are available for explicit dequeue by an application. This example does not create the application that dequeues these row LCRs.

This example illustrates a procedure DML handler that inserts records of deleted employees into an emp_del table in the hr schema. This example assumes that the emp_del table is used to retain the records of all deleted employees. The procedure DML handler is used to determine whether each row LCR contains a DELETE statement. When the procedure DML handler finds a row LCR containing a DELETE statement, it converts the DELETE into an INSERT on the emp_del table and then inserts the row.

Figure 4-1 provides an overview of the environment.
See Also:

*Oracle Streams Concepts and Administration*

### 4.2 Prerequisites

The following prerequisites must be completed before you begin the example in this chapter.

- Optionally set the `STREAMS_POOL_SIZE` initialization parameter to an appropriate value. This parameter specifies the size of the Oracle Streams pool. The Oracle Streams pool stores messages in a buffered queue and is used for internal communications during parallel capture and apply. When the `MEMORY_TARGET`, `MEMORY_MAX_TARGET`, or `SGA_TARGET` initialization parameter is set to a nonzero value, the Oracle Streams pool size is managed automatically.

  See Also:

  *Oracle Streams Replication Administrator’s Guide* for information about setting initialization parameters that are relevant to Oracle Streams

- Set the database to run in `ARCHIVELOG` mode. Any database producing changes that will be captured must run in `ARCHIVELOG` mode.
4.3 Set Up the Environment

Complete the following steps to create the `hr.emp_del` table, set up the Oracle Streams administrator, and create the queue.

1. Set Up the Environment
2. Create the `hr.emp_del` Table
3. Grant Additional Privileges to the Oracle Streams Administrator
4. Create the ANYDATA Queue at cpap.example.com
5. Check the Spool Results

**Note:**

If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line after this note to the next "END OF SCRIPT" line into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to the database.

```sql
/******************** BEGINNING OF SCRIPT ********************
```
Show Output and Spool Results
Run \texttt{SET ECHO ON} and specify the spool file for the script. Check the spool file for errors after you run this script.

\*

\texttt{SET ECHO ON}
\texttt{SPOOL streams_setup_capapp.out}

\*

Create the \texttt{hr.emp_del} Table
Connect to \texttt{cpap.example.com} as the \texttt{hr} user.

\*

\texttt{CONNECT hr@cpap.example.com}

\*

Create the \texttt{hr.emp_del} table. The columns in the \texttt{emp_del} table is the same as the columns in the \texttt{employees} table, except for one added timestamp column that will record the date when a row is inserted into the \texttt{emp_del} table.

\*

\texttt{CREATE TABLE emp_del(}
  employee_id    NUMBER(6),
  first_name     VARCHAR2(20),
  last_name      VARCHAR2(25),
  email          VARCHAR2(25),
  phone_number   VARCHAR2(20),
  hire_date      DATE,
  job_id         VARCHAR2(20),
  salary         NUMBER(8,2),
  commission_pct NUMBER(2,2),
  manager_id     NUMBER(6),
  department_id  NUMBER(4),
  timestamp      DATE);

\texttt{CREATE UNIQUE INDEX emp_del_id_pk ON emp_del (employee_id);}

\texttt{ALTER TABLE emp_del ADD (CONSTRAINT emp_del_id_pk PRIMARY KEY (employee_id));}

\*

Grant Additional Privileges to the Oracle Streams Administrator
Connect to \texttt{cpap.example.com} as \texttt{SYSTEM} user.

\*

\texttt{CONNECT SYSTEM@cpap.example.com}

\*

Grant the Oracle Streams administrator all privileges on the \texttt{emp_del} table, because the Oracle Streams administrator will be the apply user and must be able to insert records into this table. Alternatively, you can alter the apply process to specify that \texttt{hr} is the apply user.
GRANT ALL ON hr.emp_del TO STRMADMIN;

Create the ANYDATA Queue at cpap.example.com
Connect to cpap.example.com as the strmadmin user.

CONNECT strmadmin@cpap.example.com

Run the SET_UP_QUEUE procedure to create a queue named streams_queue at cpap.example.com. This queue is an ANYDATA queue that will stage the captured changes to be dequeued by an apply process and the user-constructed changes to be dequeued by a dequeue procedure. Running the SET_UP_QUEUE procedure performs the following actions:

- Creates a queue table named streams_queue_table. This queue table is owned by the Oracle Streams administrator (strmadmin) and uses the default storage of this user.
- Creates a queue named streams_queue owned by the Oracle Streams administrator (strmadmin).
- Starts the queue.

BEGIN
   DBMS_STREAMS_ADM.SET_UP_QUEUE(
      queue_table  => 'strmadmin.streams_queue_table',
      queue_name   => 'strmadmin.streams_queue');
END;
/

Check the Spool Results
Check the streams_setup_capapp.out spool file to ensure that all actions finished successfully after this script is completed.

SET ECHO OFF
SPOOL OFF

/******************************************************************** END OF SCRIPT ********************************************************************/
2. Configure the Capture Process at cpap.example.com
3. Set the Instantiation SCN for the hr.employees Table
4. Create the Procedure DML Handler handler Procedure
5. Set the Procedure DML Handler for the hr.employees Table
6. Create a Messaging Client for the Queue
7. Configure the Apply Process at cpap.example.com
8. Create a Procedure to Dequeue the Messages
9. Start the Apply Process at cpap.example.com
10. Start the Capture Process at cpap.example.com
11. Check the Spool Results

---

**Note:**

If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line after this note to the next "END OF SCRIPT" line into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect the database.

```sql
/*BEGINNING OF SCRIPT*/

Show Output and Spool Results
Run SET ECHO ON and specify the spool file for the script. Check the spool file for errors after you run this script.
*/

SET ECHO ON
SPOOL streams_config_capapp.out

/*

Configure the Capture Process at cpap.example.com
Connect to cpap.example.com as the strmadmin user.
*/

CONNECT strmadmin@cpap.example.com

/*

Configure the capture process to capture DML changes to the hr.employees table at cpap.example.com. This step creates the capture process and adds a rule to its positive rule set that instructs the capture process to capture DML changes to this table. This step also prepares the hr.employees table for instantiation and enables supplemental logging for any primary key, unique key, bitmap index, and foreign key columns in the table.
*/
```
Supplemental logging places additional information in the redo log for changes made to tables. The apply process needs this extra information to perform some operations, such as unique row identification.

See Also:

Oracle Streams Replication Administrator's Guide

*/

BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name => 'hr.employees',
    streams_type => 'capture',
    streams_name => 'capture_emp',
    queue_name => 'strmadmin.streams_queue',
    include_dml => TRUE,
    include_ddl => FALSE,
    inclusion_rule => TRUE);
END;
/
/*

Set the Instantiation SCN for the hr.employees Table
Because this example captures and applies changes in a single database, no instantiation is necessary. However, the apply process at the cpap.example.com database still must be instructed to apply changes that were made to the hr.employees table after a specific system change number (SCN).
This example uses the GET_SYSTEM_CHANGE_NUMBER function in the DBMS_FLASHBACK package to obtain the current SCN for the database. This SCN is used to run the SET_TABLE_INSTANTIATION_SCN procedure in the DBMS_APPLY_ADM package.
The SET_TABLE_INSTANTIATION_SCN procedure controls which LCRs for a table are ignored by an apply process and which LCRs for a table are applied by an apply process. If the commit SCN of an LCR for a table from a source database is less than or equal to the instantiation SCN for that table at a destination database, then the apply process at the destination database discards the LCR. Otherwise, the apply process applies the LCR. In this example, the cpap.example.com database is both the source database and the destination database.
The apply process will apply transactions to the hr.employees table with SCNs that were committed after SCN obtained in this step.

Note:

The hr.employees table also must be prepared for instantiation. This preparation was done automatically when the capture process was configured with a rule to capture DML changes to the hr.employees table in Step Configure the Capture Process at cpap.example.com.

*/
DECLARE
  iscn  NUMBER;         -- Variable to hold instantiation SCN value
BEGIN
  iscn := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER();
  DBMS_APPLY_ADM.SET_TABLE_INSTANTIATION_SCN(
    source_object_name    => 'hr.employees',
    source_database_name  => 'cpap.example.com',
    instantiation_scn     => iscn);
END;
/

CREATE OR REPLACE PROCEDURE emp_dml_handler(in_any IN ANYDATA) IS
  lcr          SYS.LCR$_ROW_RECORD;
  rc           PLS_INTEGER;
  command      VARCHAR2(30);
  old_values   SYS.LCR$_ROW_LIST;
BEGIN
  -- Access the LCR
  rc := in_any.GETOBJECT(lcr);
  -- Get the object command type
  command := lcr.GET_COMMAND_TYPE();
  -- Check for DELETE command on the hr.employees table
  IF command = 'DELETE' THEN
    -- Set the command_type in the row LCR to INSERT
    lcr.SET_COMMAND_TYPE('INSERT');
    -- Set the object name in the row LCR to EMP_DEL
    lcr.SET_OBJECT_NAME('EMP_DEL');
    -- Get the old values in the row LCR
    old_values := lcr.GET_VALUES('old');
    -- Set the old values in the row LCR to the new values in the row LCR
    lcr.SET_VALUES('new', old_values);
    -- Set the old values in the row LCR to NULL
    lcr.SET_VALUES('old', NULL);
    -- Add a SYSDATE value for the timestamp column
    lcr.ADD_COLUMN('new', 'TIMESTAMP', ANYDATA.ConvertDate(SYSDATE));
    -- Apply the row LCR as an INSERT into the hr.emp_del table
    lcr.EXECUTE(TRUE);
  END IF;
END;
/

Set the Procedure DML Handler for the hr.employees Table
Set the procedure DML handler for the hr.employees table to the procedure created in Step Create the Procedure DML Handler handler Procedure. Notice that the operation_name parameter is set to DEFAULT so that the procedure DML handler is used for each possible operation on the table, including INSERT, UPDATE, and DELETE.
BEGIN
  DBMS_APPLY_ADM.SET_DML_HANDLER(
    object_name => 'hr.employees',
    object_type => 'TABLE',
    operation_name => 'DEFAULT',
    error_handler => FALSE,
    user_procedure => 'strmadmin.emp_dml_handler',
    apply_database_link => NULL,
    apply_name => NULL);
END;
/

/*

Create a Messaging Client for the Queue
Create a messaging client that can be used by an application to dequeue the
reenqueued messages. A messaging client must be specified before the messages
can be reenqueued into the queue.
*/

BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name => 'hr.employees',
    streams_type => 'dequeue',
    streams_name => 'hr',
    queue_name => 'strmadmin.streams_queue',
    include_dml => TRUE,
    include_ddl => FALSE,
    inclusion_rule => TRUE);
END;
/

/*

Configure the Apply Process at cpap.example.com
Create an apply process to apply DML changes to the hr.employees table. Although
the procedure DML handler for the apply process causes deleted employees to be
inserted into the emp_del table, this rule specifies the employees table, because the row
LCRs in the queue contain changes to the employees table, not the emp_del table.
When you run the ADD_TABLE_RULES procedure to create the apply process, the out
parameter dml_rule_name contains the name of the DML rule created. This rule name
is then passed to the SET_ENQUEUE_DESTINATION procedure.
The SET_ENQUEUE_DESTINATION procedure in the DBMS_APPLY_ADM package specifies that
any apply process using the DML rule generated by ADD_TABLE_RULES will enqueue
messages that satisfy this rule into streams_queue. In this case, the DML rule is for row
LCRs with DML changes to the hr.employees table. A local queue other than the apply
process queue can be specified if appropriate.
*/

DECLARE
  emp_rule_name_dml VARCHAR2(30);
  emp_rule_name_ddl VARCHAR2(30);
BEGIN

Create a Procedure to Dequeue the Messages

The `emp_dq` procedure created in this step can be used to dequeue the messages that are reenqueued by the apply process. In Step Configure the Apply Process at `cpap.example.com`, the `SET_ENQUEUE_DESTINATION` procedure was used to instruct the apply process to enqueue row LCRs containing changes to the `hr.employees` table into `streams_queue`. When the `emp_dq` procedure is executed, it dequeues each row LCR in the queue and displays the type of command in the row LCR, either `INSERT`, `UPDATE`, or `DELETE`. Any information in the row LCRs can be accessed and displayed, not just the command type.

> See Also:

*Oracle Streams Concepts and Administration* for more information about displaying information in LCRs
navigation := 'NEXT MESSAGE';
COMMIT;
EXCEPTION WHEN SYS.DBMS_STREAMS_MESSAGING.ENDOFCURTRANS THEN
    navigation := 'NEXT TRANSACTION';
    WHEN DBMS_STREAMS_MESSAGING.NOMOREMSG THEN
        more_messages := FALSE;
        DBMS_OUTPUT.PUT_LINE('No more messages.');
    WHEN OTHERS THEN
        RAISE;
END;
END LOOP;
END;
/

Start the Apply Process at cpap.example.com
Set the disable_on_error parameter to n so that the apply process will not be disabled if it encounters an error, and start the apply process at cpap.example.com.

*/
BEGIN
    DBMS_APPLY_ADM.SET_PARAMETER(
        apply_name  => 'apply_emp',
        parameter   => 'disable_on_error',
        value       => 'N');
END;
/

BEGIN
    DBMS_APPLY_ADM.START_APPLY(
        apply_name  => 'apply_emp');
END;
/

Start the Capture Process at cpap.example.com
Start the capture process at cpap.example.com.

*/
BEGIN
    DBMS_CAPTURE_ADM.START_CAPTURE(
        capture_name  => 'capture_emp');
END;
/

Check the Spool Results
Check the streams_config_capapp.out spool file to ensure that all actions finished successfully after this script is completed.

*/

SET ECHO OFF
SPOOL OFF
4.5 Make DML Changes, Query for Results, and Dequeue Messages

Complete the following steps to confirm that apply process is configured correctly, make DML changes to the hr.employees table, query for the resulting inserts into the hr.emp_del table and the reenqueued messages in the streams_queue_table, and dequeue the messages that were reenqueued by the apply process:

1. Confirm the Rule Action Context
2. Perform an INSERT, UPDATE, and DELETE on hr.employees
3. Query the hr.emp_del Table and the streams_queue_table
4. Dequeue Messages Reenqueued by the Procedure DML Handler

Confirm the Rule Action Context
Step Configure the Apply Process at cpap.example.com creates an apply process rule that specifies a destination queue into which LCRs that satisfy the rule are enqueued. In this case, LCRs that satisfy the rule are row LCRs with changes to the hr.employees table.

Complete the following steps to confirm that the rule specifies a destination queue:

1. Run the following query to determine the name of the rule for DML changes to the hr.employees table used by the apply process apply_emp:

   ```sql
   CONNECT strmadmin@cpap.example.com
   Enter password: password

   SELECT RULE_OWNER, RULE_NAME FROM DBA_STREAMS_RULES
   WHERE STREAMS_NAME = 'APPLY_EMP' AND
     STREAMS_TYPE = 'APPLY' AND
     SCHEMA_NAME  = 'HR' AND
     OBJECT_NAME  = 'EMPLOYEES' AND
     RULE_TYPE    = 'DML'
   ORDER BY RULE_NAME;
   
   Your output looks similar to the following:
   
<table>
<thead>
<tr>
<th>RULE_OWNER</th>
<th>RULE_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRMADMIN</td>
<td>EMPLOYEES3</td>
</tr>
</tbody>
</table>
   
   2. View the action context for the rule returned by the query in Step 1:

   ```sql
   COLUMN RULE_OWNER HEADING 'Rule Owner' FORMAT A15
   COLUMN DESTINATION_QUEUE_NAME HEADING 'Destination Queue' FORMAT A30

   SELECT RULE_OWNER, DESTINATION_QUEUE_NAME
   FROM DBA_APPLY_ENQUEUE
   WHERE RULE_NAME = 'EMPLOYEES3'
   ORDER BY DESTINATION_QUEUE_NAME;
   ```
Ensure that you substitute the rule name returned in Step 1 in the WHERE clause. Your output looks similar to the following:

<table>
<thead>
<tr>
<th>Rule Owner</th>
<th>Destination Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRMADMIN</td>
<td>&quot;STRMADMIN&quot;.&quot;STREAMS_QUEUE&quot;</td>
</tr>
</tbody>
</table>

The output should show that LCRs that satisfy the apply process rule are enqueued into streams_queue.

**Perform an INSERT, UPDATE, and DELETE on hr.employees**

Make the following DML changes to the hr.employees table.

```
CONNECT hr@cpap.example.com
Enter password: password

INSERT INTO hr.employees VALUES(207, 'JOHN', 'SMITH', 'JSMITH@EXAMPLE.COM', NULL, '07-JUN-94', 'AC_ACCOUNT', 777, NULL, NULL, 110);
COMMIT;

UPDATE hr.employees SET salary=5999 WHERE employee_id=207;
COMMIT;

DELETE FROM hr.employees WHERE employee_id=207;
COMMIT;
```

**Query the hr.emp_del Table and the streams_queue_table**

After some time passes to allow for capture and apply of the changes performed in the previous step, run the following queries to see the results:

```
CONNECT strmadmin@cpap.example.com
Enter password: password

SELECT employee_id, first_name, last_name, timestamp
FROM hr.emp_del ORDER BY employee_id;

SELECT MSG_ID, MSG_STATE, CONSUMER_NAME
FROM AQ$STREAMS_QUEUE_TABLE ORDER BY MSG_ID;
```

When you run the first query, you should see a record for the employee with an employee_id of 207. This employee was deleted in the previous step. When you run the second query, you should see the reenqueued messages resulting from all of the changes in the previous step, and the MSG_STATE should be READY for these messages.

**Dequeue Messages Reenqueued by the Procedure DML Handler**

Use the emp_dq procedure to dequeue the messages that were reenqueued by the procedure DML handler.

```
SET SERVEROUTPUT ON SIZE 100000
EXEC emp_dq('HR');
```

For each row changed by a DML statement, one line is returned, and each line states the command type of the change (either INSERT, UPDATE, or DELETE). If you repeat the query on the queue table in Step Query the hr.emp_del Table and the streams_queue_table after the messages are dequeued, then the dequeued messages should have been consumed. That is, either the MSG_STATE should be PROCESSED for these messages, or the messages should no longer be in the queue.
SELECT MSG_ID, MSG_STATE, CONSUMER_NAME
    FROM AQ$STREAMS_QUEUE_TABLE ORDER BY MSG_ID;
Logical Change Records with LOBs

Example

This chapter illustrates an example that creates a PL/SQL procedure for constructing and enqueuing LCRs that contain LOBs.

This chapter contains this topic:

- Example Script for Constructing and Enqueuing LCRs Containing LOBs

5.1 Example Script for Constructing and Enqueuing LCRs Containing LOBs

1. Show Output and Spool Results
2. Grant the Oracle Streams Administrator EXECUTE Privilege on DBMS_STREAMS_MESSAGING
3. Connect as the Oracle Streams Administrator
4. Create an ANYDATA Queue
5. Create and Start an Apply Process
6. Create a Schema with Tables Containing LOB Columns
7. Grant the Oracle Streams Administrator Necessary Privileges on the Tables
8. Create a PL/SQL Procedure to Enqueue LCRs Containing LOBs
9. Create the do_enq_clob Function to Enqueue CLOB Data
10. Enqueue CLOB Data Using the do_enq_clob Function
11. Check the Spool Results

Note:

If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line after this note to the next "END OF SCRIPT" line into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to all of the databases in the environment.

/******************** BEGINNING OF SCRIPT ********************/
Show Output and Spool Results
Run SET ECHO ON and specify the spool file for the script. Check the spool file for errors after you run this script.

*/

SET ECHO ON
SPOOL lob_construct.out

/*

Grant the Oracle Streams Administrator EXECUTE Privilege on DBMS_STREAMS_MESSAGING
Explicit EXECUTE privilege on the package is required because a procedure in the package is called in within a PL/SQL procedure in Step Create a PL/SQL Procedure to Enqueue LCRs Containing LOBs.

*/

CONNECT / AS SYSDBA;

GRANT EXECUTE ON DBMS_STREAMS_MESSAGING TO strmadmin;

/*

Connect as the Oracle Streams Administrator
*/

SET ECHO ON
SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 100
SET SERVEROUTPUT ON SIZE 100000

CONNECT strmadmin

/*

Create an ANYDATA Queue
*/

BEGIN
   DBMS_STREAMS_ADM.SET_UP_QUEUE(
      queue_table => 'lobex_queue_table',
      queue_name => 'lobex_queue');
END;
/
/*

Create and Start an Apply Process
*/
BEGIN
    DBMS_APPLY_ADM.CREATE_APPLY(
        queue_name      => 'strmadmin.lobex_queue',
        apply_name      => 'apply_lob',
        apply_captured  => FALSE);
END;
/

BEGIN
    DBMS_APPLY_ADM.SET_PARAMETER(
        apply_name => 'apply_lob',
        parameter  => 'disable_on_error',
        value      => 'N');
END;
/

BEGIN
    DBMS_APPLY_ADM.START_APPLY(
        'apply_lob');
END;
/

/*

Create a Schema with Tables Containing LOB Columns
*/

CONNECT system

CREATE TABLESPACE lob_user_tbs DATAFILE 'lob_user_tbs.dbf'
    SIZE 5M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;

ACCEPT password PROMPT 'Enter password for user: ' HIDE

CREATE USER lob_user
    IDENTIFIED BY &password
    DEFAULT TABLESPACE lob_user_tbs
    QUOTA UNLIMITED ON lob_user_tbs;

GRANT ALTER SESSION, CREATE CLUSTER, CREATE DATABASE LINK, CREATE SEQUENCE,
    CREATE SESSION, CREATE SYNONYM, CREATE TABLE, CREATE VIEW, CREATE INDEXTYPE,
    CREATE OPERATOR, CREATE PROCEDURE, CREATE TRIGGER, CREATE TYPE
    TO lob_user;

CONNECT lob_user/lob_user_pw

CREATE TABLE with_clob (a NUMBER PRIMARY KEY,
    c1 CLOB,
    c2 CLOB,
    c3 CLOB);

CREATE TABLE with_blob (a NUMBER PRIMARY KEY,
    b BLOB);

/*
Grant the Oracle Streams Administrator Necessary Privileges on the Tables

Granting these privileges enables the Oracle Streams administrator to get the LOB length for offset and to perform DML operations on the tables.

*/

GRANT ALL ON with_clob TO strmadmin;
GRANT ALL ON with_blob TO strmadmin;
COMMIT;

/ *

Create a PL/SQL Procedure to Enqueue LCRs Containing LOBs

*/

CONNECT strmadmin

CREATE OR REPLACE PROCEDURE enq_row_lcr(source_dbname VARCHAR2,
   cmd_type VARCHAR2,
   obj_owner VARCHAR2,
   obj_name VARCHAR2,
   old_vals SYS.LCR$_ROW_LIST,
   new_vals SYS.LCR$_ROW_LIST) AS

   xr_lcr SYS.LCR$_ROW_RECORD;

BEGIN
   xr_lcr := SYS.LCR$_ROW_RECORD.CONSTRUCT(
                source_database_name => source_dbname,
                command_type         => cmd_type,
                object_owner         => obj_owner,
                object_name          => obj_name,
                old_values           => old_vals,
                new_values           => new_vals);

   -- Enqueue a row lcr
   DBMS_STREAMS_MESSAGING.ENQUEUE(
       queue_name => 'lobex_queue',
       payload    => ANYDATA.ConvertObject(xr_lcr));

END enq_row_lcr;
/
SHOW ERRORS

/ *

Create the do_enq_clob Function to Enqueue CLOB Data

*/

-- Description of each variable:
-- src_dbname : Source database name
-- tab_owner  : Table owner
-- tab_name   : Table name
-- col_name   : Name of the CLOB column
-- new_vals   : SYS.LCR$_ROW_LIST containing primary key and supplementally
--             logged columns
-- clob_data  : CLOB that contains data to be sent
-- offset     : Offset from which data should be sent, default is 1
-- lsize      : Size of data to be sent, default is 0
-- chunk_size : Size used for creating LOB chunks, default is 2048
CREATE OR REPLACE FUNCTION do_enq_clob(src_dbname VARCHAR2, 
tab_owner VARCHAR2, 
tab_name VARCHAR2, 
col_name VARCHAR2, 
new_vals SYS.LCR$_ROW_LIST, 
clob_data CLOB, 
offset NUMBER default 1, 
lsize NUMBER default 0, 
chunk_size NUMBER default 2048)
RETURN NUMBER IS
lob_offset NUMBER; -- maintain lob offset
newunit SYS.LCR$_ROW_UNIT;
tnewvals SYS.LCR$_ROW_LIST;
lob_flag NUMBER;
lob_data VARCHAR2(32767);
lob_size NUMBER;
unit_pos NUMBER;
final_size NUMBER;
exit_flg BOOLEAN;
c_size NUMBER;
i NUMBER;
BEGIN
lob_size := DBMS_LOB.GETLENGTH(clob_data);
unit_pos := new_vals.count + 1;
tnewvals := new_vals;
c_size := chunk_size;
i := 0;
-- validate parameters
IF (unit_pos <= 1) THEN
DBMS_OUTPUT.PUT_LINE('Invalid new_vals list');
RETURN 1;
END IF;

IF (c_size < 1) THEN
DBMS_OUTPUT.PUT_LINE('Invalid LOB chunk size');
RETURN 1;
END IF;

IF (lsize < 0 OR lsize > lob_size) THEN
DBMS_OUTPUT.PUT_LINE('Invalid LOB size');
RETURN 1;
END IF;

IF (offset < 1 OR offset >= lob_size) THEN
DBMS_OUTPUT.PUT_LINE('Invalid lob offset');
RETURN 1;
ELSE
lob_offset := offset;
END IF;

-- calculate final size
IF (lsize = 0) THEN
final_size := lob_size;
ELSE
final_size := lob_offset + lsize;
END IF;

-- The following output lines are for debugging purposes only.
-- DBMS_OUTPUT.PUT_LINE('Final size: ' || final_size);
-- DBMS_OUTPUT.PUT_LINE('Lob size: ' || lob_size);

IF (final_size < 1 OR final_size > lob_size) THEN
    DBMS_OUTPUT.PUT_LINE('Invalid lob size');
    RETURN 1;
END IF;

-- expand new_vals list for LOB column
tnewvals.extend();

exit_flg := FALSE;

-- Enqueue all LOB chunks
LOOP
    -- The following output line is for debugging purposes only.
    DBMS_OUTPUT.PUT_LINE('About to write chunk#' || i);
    i := i + 1;

    -- check if last LOB chunk
    IF ((lob_offset + c_size) < final_size) THEN
        lob_flag := DBMS_LCR.LOB_CHUNK;
    ELSE
        lob_flag := DBMS_LCR.LAST_LOB_CHUNK;
        exit_flg := TRUE;
    END IF;
    -- The following output line is for debugging purposes only.
    DBMS_OUTPUT.PUT_LINE('Last LOB chunk');
    END IF;

    -- The following output lines are for debugging purposes only.
    DBMS_OUTPUT.PUT_LINE('lob offset: ' || lob_offset);
    DBMS_OUTPUT.PUT_LINE('Chunk size: ' || to_char(c_size));

    lob_data := DBMS_LOB.SUBSTR(clob_data, c_size, lob_offset);

    -- create row unit for clob
    newunit := SYS.LCR$_ROW_UNIT(col_name,
                              ANYDATA.ConvertVarChar2(lob_data),
                              lob_flag,
                              lob_offset,
                              NULL);

    -- insert new LCR$_ROW_UNIT
tenewvals(unit_pos) := newunit;

    -- enqueue lcr
    enq_row_lcr(
        source_dbname => src_dbname,
        cmd_type      => 'LOB WRITE',
        obj_owner     => tab_owner,
        obj_name      => tab_name,
        old_vals      => NULL,
        new_vals      => tnewvals);

    -- calculate next chunk size
    lob_offset := lob_offset + c_size;

    IF ((final_size - lob_offset) < c_size) THEN
        c_size := final_size - lob_offset + 1;
BEGIN
-- Create primary key for LCR$_ROW_UNIT
newunit1 := SYS.LCR$_ROW_UNIT('A',
    ANYDATA.ConvertNumber(3),
    NULL,
    NULL,
    NULL);

-- Create empty CLOBs
newunit2 := sys.lcr$_row_unit('C1',
    ANYDATA.ConvertVarChar2(NULL),
    DBMS_LCR.EMPTY_LOB,
    NULL,
    NULL);

newunit3 := SYS.LCR$_ROW_UNIT('C2',
    ANYDATA.ConvertVarChar2(NULL),
    DBMS_LCR.EMPTY_LOB,
    NULL,
    NULL);

newunit4 := SYS.LCR$_ROW_UNIT('C3',
    ANYDATA.ConvertVarChar2(NULL),
    DBMS_LCR.EMPTY_LOB,
    NULL,
    NULL);

END;
Chapter 5
Example Script for Constructing and Enqueuing LCRs Containing LOBs

```
DBMS_LCR.EMPTY_LOB,
NULL,
NULL);
newvals := SYS.LCR$_ROW_LIST(newunit1,newunit2,newunit3,newunit4);

-- Perform an insert
enq_row_lcr(
  source_dbname => 'MYDB.EXAMPLE.COM',
  cmd_type      => 'INSERT',
  obj_owner     => 'LOB_USER',
  obj_name      => 'WITH_CLOB',
  old_vals      => NULL,
  new_vals      => newvals);

-- construct clobs
big_data := RPAD('Hello World', 1000, '_');
big_data := big_data || '#';
big_data := big_data || big_data || big_data || big_data || big_data;
DBMS_LOB.CREATETEMPORARY(
  lob_loc => c1_data,
  cache   => TRUE);
DBMS_LOB.WRITEAPPEND(
  lob_loc => c1_data,
  amount => length(big_data),
  buffer => big_data);

big_data := RPAD('1234567890#', 1000, '_');
big_data := big_data || big_data || big_data || big_data;
DBMS_LOB.CREATETEMPORARY(
  lob_loc => c2_data,
  cache   => TRUE);
DBMS_LOB.WRITEAPPEND(
  lob_loc => c2_data,
  amount => length(big_data),
  buffer => big_data);

big_data := RPAD('ASDFGHJKLQW', 2000, '_');
big_data := big_data || '#';
big_data := big_data || big_data || big_data || big_data || big_data;
DBMS_LOB.CREATETEMPORARY(
  lob_loc => c3_data,
  cache   => TRUE);
DBMS_LOB.WRITEAPPEND(
  lob_loc => c3_data,
  amount => length(big_data),
  buffer => big_data);

-- pk info
newunit1 := SYS.LCR$_ROW_UNIT('A',
  ANYDATA.ConvertNumber(3),
  NULL,
  NULL,
  NULL);
newvals := SYS.LCR$_ROW_LIST(newunit1);

-- write c1 clob
n := do_enq_clob(
  src_dbname => 'MYDB.EXAMPLE.COM',
  tab_owner  => 'LOB_USER',
  ...)
```
tab_name => 'WITH_CLOB',
col_name => 'C1',
new_vals => newvals,
clob_data => c1_data,
offset => 1,
chunk_size => 1024);
DBMS_OUTPUT.PUT_LINE('n=' || n);

-- write c2 clob
newvals := SYS.LCR$_ROW_LIST(newunit1);
N := do_enq_clob(
    src_dbname => 'MYDB.EXAMPLE.COM',
    tab_owner => 'LOB_USER',
    tab_name => 'WITH_CLOB',
    col_name => 'C2',
    new_vals => newvals,
    clob_data => c2_data,
    offset => 1,
    chunk_size => 2000);
DBMS_OUTPUT.PUT_LINE('n=' || n);

-- write c3 clob
newvals := SYS.LCR$_ROW_LIST(newunit1);
N := do_enq_clob(src_dbname=>'MYDB.EXAMPLE.COM',
    tab_owner => 'LOB_USER',
    tab_name => 'WITH_CLOB',
    col_name => 'C3',
    new_vals => newvals,
    clob_data => c3_data,
    offset => 1,
    chunk_size => 500);
DBMS_OUTPUT.PUT_LINE('n=' || n);

COMMIT;

END;
/

/*

Check the Spool Results
Check the lob_construct.out spool file to ensure that all actions completed successfully after this script completes.
*/

SET ECHO OFF
SPool OFF

/*************************** END OF SCRIPT ******************************/

After you run the script, you can check the lob_user.with_clob table to list the rows applied by the apply process. The DBMS_LOCK.SLEEP statement is used to give the apply process time to apply the enqueued rows.

CONNECT lob_user/lob_user_pw

EXECUTE DBMS_LOCK.SLEEP(10);
SELECT a, c1, c2, c3 FROM with_clob ORDER BY a;

SELECT a, LENGTH(c1), LENGTH(c2), LENGTH(c3) FROM with_clob ORDER BY a;
Rule-Based Application Example

This chapter illustrates a rule-based application that uses the Oracle rules engine.

The examples in this chapter are independent of Oracle Streams. That is, no Oracle Streams capture processes, propagations, apply processes, or messaging clients are clients of the rules engine in these examples, and no queues are used.

The following topics describe configuring examples of rules-based applications:

- Overview of the Rule-Based Application
- Using Rules on Nontable Data Stored in Explicit Variables
- Using Rules on Data in Explicit Variables with Iterative Results
- Using Partial Evaluation of Rules on Data in Explicit Variables
- Using Rules on Data Stored in a Table
- Using Rules on Both Explicit Variables and Table Data
- Using Rules on Implicit Variables and Table Data
- Using Event Contexts and Implicit Variables with Rules
- Dispatching Problems and Checking Results for the Table Examples

See Also:
Oracle Streams Concepts and Administration

6.1 Overview of the Rule-Based Application

Each example in this chapter creates a rule-based application that handles customer problems. The application uses rules to determine actions that must be completed based on the problem priority when a new problem is reported. For example, the application assigns each problem to a particular company center based on the problem priority.

The application enforces these rules using the rules engine. An evaluation context named evalctx is created to define the information surrounding a support problem. Rules are created based on the requirements described previously, and they are added to a rule set named rs.

The task of assigning problems is done by a user-defined procedure named problem_dispatch, which calls the rules engine to evaluate rules in the rule set rs and then takes appropriate action based on the rules that evaluate to TRUE.
6.2 Using Rules on Nontable Data Stored in Explicit Variables

This example illustrates how to use rules to evaluate data stored in explicit variables. This example handles customer problems based on priority and uses the following rules for handling customer problems:

- Assign all problems with priority greater than 2 to the San Jose Center.
- Assign all problems with priority less than or equal to 2 to the New York Center.
- Send an alert to the vice president of support for a problem with priority equal to 1.

The evaluation context contains only one explicit variable named `priority`, which refers to the priority of the problem being dispatched. The value for this variable is passed to `DBMS_RULE.EVALUATE` procedure by the `problem_dispatch` procedure.

Complete the following steps:

1. Show Output and Spool Results
2. Create the support User
3. Grant the support User the Necessary System Privileges on Rules
4. Create the evalctx Evaluation Context
5. Create the Rules that Correspond to Problem Priority
6. Create the rs Rule Set
7. Add the Rules to the Rule Set
8. Query the Data Dictionary
9. Create the problem_dispatch PL/SQL Procedure
10. Dispatch Sample Problems
11. Check the Spool Results

---

**Note:**

If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line after this note to the next "END OF SCRIPT" line into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to all of the databases in the environment.

```sql
/******************** BEGINNING OF SCRIPT ********************

Show Output and Spool Results
Run SET ECHO ON and specify the spool file for the script. Check the spool file for errors after you run this script.
```
SET ECHO ON
SPOOL rules_stored_variables.out
/

Create the support User
*/

CONNECT SYSTEM

ACCEPT password PROMPT 'Enter password for user: ' HIDE

GRANT ALTER SESSION, CREATE CLUSTER, CREATE DATABASE LINK, CREATE SEQUENCE, CREATE SESSION, CREATE SYNONYM, CREATE TABLE, CREATE VIEW, CREATE INDEXTYPE, CREATE OPERATOR, CREATE PROCEDURE, CREATE TRIGGER, CREATE TYPE TO support IDENTIFIED BY &password;
/

Grant the support User the Necessary System Privileges on Rules
*/

BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege  => DBMS_RULE_ADM.CREATE_RULE_SET_OBJ,
    grantee    => 'support',
    grant_option => FALSE);
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege  => DBMS_RULE_ADM.CREATE_RULE_OBJ,
    grantee    => 'support',
    grant_option => FALSE);
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege  => DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT_OBJ,
    grantee    => 'support',
    grant_option => FALSE);
END;
/
/

Create the evalctx Evaluation Context
*/

CONNECT support

SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 100
SET SERVEROUTPUT ON
DECLARE
  vt SYS.RE$VARIABLE_TYPE_LIST;
BEGIN
  vt := SYS.RE$VARIABLE_TYPE_LIST(
    SYS.RE$VARIABLE_TYPE('priority', 'NUMBER', NULL, NULL));
  DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT(
    evaluation_context_name => 'evalctx',
    variable_types => vt,
    evaluation_context_comment => 'support problem definition');
END;
/
/

Create the Rules that Correspond to Problem Priority
The following code creates one action context for each rule, and one name-value pair in each action context.
*/

DECLARE
  ac SYS.RE$NV_LIST;
BEGIN
  ac := SYS.RE$NV_LIST(NULL);
  ac.ADD_PAIR('CENTER', ANYDATA.CONVERTVARCHAR2('San Jose'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name => 'r1',
    condition => ':priority > 2',
    action_context => ac,
    rule_comment => 'Low priority problems');
  ac := SYS.RE$NV_LIST(NULL);
  ac.ADD_PAIR('CENTER', ANYDATA.CONVERTVARCHAR2('New York'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name => 'r2',
    condition => ':priority <= 2',
    action_context => ac,
    rule_comment => 'High priority problems');
  ac := SYS.RE$NV_LIST(NULL);
  ac.ADD_PAIR('ALERT', ANYDATA.CONVERTVARCHAR2('John Doe'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name => 'r3',
    condition => ':priority = 1',
    action_context => ac,
    rule_comment => 'Urgent problems');
END;
/
/

Create the rs Rule Set
*/

BEGIN
  DBMS_RULE_ADM.CREATE_RULE_SET(
    rule_set_name => 'rs',
    evaluation_context => 'evalctx',
    rule_set_comment => 'support rules');
END;
/

/*

Add the Rules to the Rule Set
*/

BEGIN

DBMS_RULE_ADM.ADD_RULE(
  rule_name        => 'r1',
  rule_set_name    => 'rs');

DBMS_RULE_ADM.ADD_RULE(
  rule_name        => 'r2',
  rule_set_name    => 'rs');

DBMS_RULE_ADM.ADD_RULE(
  rule_name        => 'r3',
  rule_set_name    => 'rs');

END;
/

/*

Query the Data Dictionary
At this point, you can view the evaluation context, rules, and rule set you created in
the previous steps.
*/

COLUMN EVALUATION_CONTEXT_NAME HEADING 'Eval Context Name' FORMAT A30
COLUMN EVALUATION_CONTEXT_COMMENT HEADING 'Eval Context Comment' FORMAT A40

SELECT EVALUATION_CONTEXT_NAME, EVALUATION_CONTEXT_COMMENT
  FROM USER_EVALUATION_CONTEXTS
  ORDER BY EVALUATION_CONTEXT_NAME;

SET LONGCHUNKSIZE 4000
SET LONG 4000
COLUMN RULE_NAME HEADING 'Rule|Name' FORMAT A5
COLUMN RULE_CONDITION HEADING 'Rule Condition' FORMAT A35
COLUMN ACTIONCONTEXT_NAME HEADING 'Action|Context|Name' FORMAT A10
COLUMN ACTIONCONTEXT_VALUE HEADING 'Action|Context|Value' FORMAT A10

SELECT RULE_NAME,
  RULE_CONDITION,
  AC.NVN_NAME ACTIONCONTEXT_NAME,
  AC.NVN_VALUE.ACCESSVARCHAR2() ACTIONCONTEXT_VALUE
  FROM USER_RULES R, TABLE(R.RULE_ACTIONCONTEXT.ACTX_LIST) AC
  ORDER BY RULE_NAME;

COLUMN RULE_SET_NAME HEADING 'Rule Set Name' FORMAT A20
COLUMN RULE_SET_EVALCONTEXT_OWNER HEADING 'Eval Context|Owner' FORMAT A12
COLUMN RULE_SET_EVALCONTEXT_NAME HEADING 'Eval Context Name' FORMAT A25
COLUMN RULE_SET_COMMENT HEADING 'Rule Set|Comment' FORMAT A15

SELECT RULE_SET_NAME,
  RULE_SET_EVALCONTEXT_OWNER,
  RULE_SET_EVALCONTEXT_NAME,
  RULE_SET_COMMENT
FROM USER_RULE_SETS
ORDER BY RULE_SET_NAME;

/*
Create the problem_dispatch PL/SQL Procedure
*/

CREATE OR REPLACE PROCEDURE problem_dispatch (priority NUMBER) IS
    vv        SYS.RE$VARIABLE_VALUE;
    vvl       SYS.RE$VARIABLE_VALUE_LIST;
    truehits  SYS.RE$RULE_HIT_LIST;
    maybehits SYS.RE$RULE_HIT_LIST;
    ac        SYS.RE$NV_LIST;
    namearray SYS.RE$NAME_ARRAY;
    name      VARCHAR2(30);
    cval      VARCHAR2(100);
    rnum      INTEGER;
    i         INTEGER;
    status    PLS_INTEGER;
BEGIN
    vv  := SYS.RE$VARIABLE_VALUE(priority,
                                  ANYDATA.CONVERTNUMBER(priority));
    vvl := SYS.RE$VARIABLE_VALUE_LIST(vv);
    truehits := SYS.RE$RULE_HIT_LIST();
    maybehits := SYS.RE$RULE_HIT_LIST();
    DBMS_RULE.EVALUATE(
        rule_set_name        => 'support.rs',
        evaluation_context   => 'evalctx',
        variable_values      => vvl,
        true_rules           => truehits,
        maybe_rules          => maybehits);
    FOR rnum IN 1..truehits.COUNT LOOP
        DBMS_OUTPUT.PUT_LINE('Using rule ' || truehits(rnum).rule_name);
        ac := truehits(rnum).rule_action_context;
        namearray := ac.GET_ALL_NAMES;
        FOR i IN 1..namearray.count loop
            name := namearray(i);
            status := ac.GET_VALUE(name).GETVARCHAR2(cval);
            IF (name = 'CENTER') then
                DBMS_OUTPUT.PUT_LINE('Assigning problem to ' || cval);
            ELSIF (name = 'ALERT') THEN
                DBMS_OUTPUT.PUT_LINE('Sending alert to: ' || cval);
            END IF;
        END LOOP;
    END LOOP;
END;
/

/*
Dispatch Sample Problems
*/

EXECUTE problem_dispatch(1);
EXECUTE problem_dispatch(2);
EXECUTE problem_dispatch(3);
EXECUTE problem_dispatch(5);

/

Check the Spool Results
Check the rules_stored_variables.out spool file to ensure that all actions completed successfully after this script completes.

SET ECHO OFF
SPOOL OFF

/**************************** END OF SCRIPT ******************************/

6.3 Using Rules on Data in Explicit Variables with Iterative Results

This example is the same as the previous example "Using Rules on Nontable Data Stored in Explicit Variables", except that this example returns evaluation results iteratively instead of all at once.

Complete the following steps:

1. Show Output and Spool Results
2. Ensure That You Have Completed the Preliminary Steps
3. Replace the problem_dispatch PL/SQL Procedure
4. Dispatch Sample Problems
5. Clean Up the Environment (Optional)
6. Check the Spool Results

Note:

If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line after this note to the next "END OF SCRIPT" line into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to all of the databases in the environment.

/**************************** BEGINNING OF SCRIPT ******************************/

Show Output and Spool Results
Run SET ECHO ON and specify the spool file for the script. Check the spool file for errors after you run this script.

*/
Ensure That You Have Completed the Preliminary Steps

Ensure that you have completed Steps Show Output and Spool Results to Query the Data Dictionary in the "Using Rules on Nontable Data Stored in Explicit Variables". If you have not completed these steps, then complete them before you continue.

Replace the problem_dispatch PL/SQL Procedure

Replace the problem_dispatch procedure created in Step Create the problem_dispatch PL/SQL Procedure with the procedure in this step. The difference between the two procedures is that the procedure created in Step Create the problem_dispatch PL/SQL Procedure returns all evaluation results at once while the procedure in this step returns evaluation results iteratively.

CONNECT support

SET SERVEROUTPUT ON
CREATE OR REPLACE PROCEDURE problem_dispatch (priority NUMBER)
IS
vv SYS.RE$VARIABLE_VALUE;
vvl SYS.RE$VARIABLE_VALUE_LIST;
truehits BINARY_INTEGER;
maybehits BINARY_INTEGER;
hit SYS.RE$RULE_HIT;
ac SYS.RE$NV_LIST;
namearray SYS.RE$NAME_ARRAY;
name VARCHAR2(30);
cval VARCHAR2(100);
i INTEGER;
status PLS_INTEGER;
iter_closed EXCEPTION;
pragma exception_init(iter_closed, -25453);
BEGIN
vv := SYS.RE$VARIABLE_VALUE('priority',
ANYDATA.CONVERTNUMBER(priority));
vvl := SYS.RE$VARIABLE_VALUE_LIST(vv);
DBMS_RULE.EVALUATE{
    rule_set_name => 'support.rs',
    evaluation_context => 'evalctx',
    variable_values => vvl,
    true_rules_iterator => truehits,
    maybe_rules_iterator => maybehits};
LOOP
hit := DBMS_RULE.GET_NEXT_HIT(truehits);
EXIT WHEN hit IS NULL;
DBMS_OUTPUT.PUT_LINE('Using rule ' || hit.rule_name);
ac := hit.rule_action_context;
namearray := ac.GET_ALL_NAMES;
FOR i IN 1..namearray.COUNT LOOP
    name := namearray(i);
    status := ac.GET_VALUE(name).GETVARCHAR2(cval);
    IF (name = 'CENTER') then
        DBMS_OUTPUT.PUT_LINE('Assigning problem to ' || cval);
    ELSIF (name = 'ALERT') THEN
        DBMS_OUTPUT.PUT_LINE('Sending alert to: '|| cval);
    END IF;
END LOOP;
END LOOP;
-- Close iterators
BEGIN
    DBMS_RULE.CLOSE_ITERATOR(truehits);
EXCEPTION
    WHEN iter_closed THEN
        NULL;
END;
BEGIN
    DBMS_RULE.CLOSE_ITERATOR(maybehits);
EXCEPTION
    WHEN iter_closed THEN
        NULL;
END;
END;
/*
Dispatch Sample Problems
*/
EXECUTE problem_dispatch(1);
EXECUTE problem_dispatch(2);
EXECUTE problem_dispatch(3);
EXECUTE problem_dispatch(5);
/*
Clean Up the Environment (Optional)
You can clean up the sample environment by dropping the support user.
*/
CONNECT SYSTEM
DROP USER support CASCADE;
/*
Check the Spool Results
Check the rules_stored_variables_iterative.out spool file to ensure that all actions completed successfully after this script completes.
*/
SET ECHO OFF
SPOOL OFF
/*******************************************************************************/
6.4 Using Partial Evaluation of Rules on Data in Explicit Variables

This example illustrates how to use partial evaluation when an event causes rules to evaluate to MAYBE instead of TRUE or FALSE. This example handles customer problems based on priority and problem type, and uses the following rules for handling customer problems:

- Assign all problems whose problem type is HARDWARE to the San Jose Center.
- Assign all problems whose problem type is SOFTWARE to the New York Center.
- Assign all problems whose problem type is NULL (unknown) to the Texas Center.
- Send an alert to the vice president of support for a problem with priority equal to 1.

Problems whose problem type is NULL evaluate to MAYBE. This example uses partial evaluation to take an action when MAYBE rules are returned to the rules engine client. In this case, the action is to assign the problem to the Texas Center.

The evaluation context contains an explicit variable named priority, which refers to the priority of the problem being dispatched. The evaluation context also contains an explicit variable named problem_type, which refers to the type of problem being dispatched (either HARDWARE or SOFTWARE). The values for these variables are passed to DBMS_RULE.EVALUATE procedure by the problem_dispatch procedure.

Complete the following steps:

1. Show Output and Spool Results
2. Create the support User
3. Grant the support User the Necessary System Privileges on Rules
4. Create the evalctx Evaluation Context
5. Create the Rules that Correspond to Problem Priority
6. Create the rs Rule Set
7. Add the Rules to the Rule Set
8. Query the Data Dictionary
9. Create the problem_dispatch PL/SQL Procedure
10. Dispatch Sample Problems
11. Clean Up the Environment (Optional)
12. Check the Spool Results
Show Output and Spool Results
Run `SET ECHO ON` and specify the spool file for the script. Check the spool file for errors after you run this script.

```sql
SET ECHO ON
SPOOL rules_stored_variables_partial.out
```

Create the support User

```sql
CONNECT SYSTEM
ACCEPT password PROMPT 'Enter password for user: ' 
GRANT ALTER SESSION, CREATE CLUSTER, CREATE DATABASE LINK, CREATE SEQUENCE,
   CREATE SESSION, CREATE SYNONYM, CREATE TABLE, CREATE VIEW, CREATE INDEXYPE,
   CREATE OPERATOR, CREATE PROCEDURE, CREATE TRIGGER, CREATE TYPE
TO support IDENTIFIED BY &password;
```

Grant the support User the Necessary System Privileges on Rules

```sql
BEGIN
   DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
      privilege => DBMS_RULE_ADM.CREATE_RULE_SET_OBJ,
      grantee   => 'support',
      grant_option => FALSE);
   DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
      privilege => DBMS_RULE_ADM.CREATE_RULE_OBJ,
      grantee   => 'support',
      grant_option => FALSE);
   DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
      privilege => DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT_OBJ,
      grantee   => 'support',
      grant_option => FALSE);
END;
```
CREATE the evalctx Evaluation Context

CONNECT support

SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 100
SET SERVEROUTPUT ON
DECLARE
  vt SYS.RE$VARIABLE_TYPE_LIST;
BEGIN
  vt := SYS.RE$VARIABLE_TYPE_LIST(
    SYS.RE$VARIABLE_TYPE('priority', 'NUMBER', NULL, NULL),
    SYS.RE$VARIABLE_TYPE('problem_type', 'VARCHAR2(30)', NULL, NULL));
  DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT(
    evaluation_context_name => 'evalctx',
    variable_types => vt,
    evaluation_context_comment => 'support problem definition');
end;
/

CREATE the Rules that Correspond to Problem Priority
The following code creates one action context for each rule, and one name-value pair in each action context.

DECLARE
  ac SYS.RE$NV_LIST;
begin
  ac := SYS.RE$NV_LIST(NULL);
  ac.ADD_PAIR('ALERT', ANYDATA.CONVERTVARCHAR2('John Doe'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name => 'r1',
    condition => ':priority = 1',
    action_context => ac,
    rule_comment => 'Urgent problems');
  ac := SYS.RE$NV_LIST(NULL);
  ac.ADD_PAIR('TRUE CENTER', ANYDATA.CONVERTVARCHAR2('San Jose'));
  DBMS_RULE_ADM.CREATE_ACTION_CONTEXT(
    rule_name => 'r2',
    condition => ':problem_type = ' HARDWARE '',
    action_context => ac,
    rule_comment => 'Hardware problems');
  ac := SYS.RE$NV_LIST(NULL);
  ac.ADD_PAIR('TRUE CENTER', ANYDATA.CONVERTVARCHAR2('New York'));
  ac.ADD_PAIR('MAYBE CENTER', ANYDATA.CONVERTVARCHAR2('Texas'));

DBMS_RULE_ADM.CREATE_RULE(
  rule_name       => 'r3',
  condition       => ':problem_type = ''SOFTWARE''',
  action_context  => ac,
  rule_comment    => 'Software problems');
END;
/
/
/*
Create the rs Rule Set
*/
BEGIN
  DBMS_RULE_ADM.CREATE_RULE_SET(
    rule_set_name      => 'rs',
    evaluation_context => 'evalctx',
    rule_set_comment   => 'support rules');
END;
/
/
/*
Add the Rules to the Rule Set
*/
BEGIN
  DBMS_RULE_ADM.ADD_RULE(
    rule_name     => 'r1',
    rule_set_name => 'rs');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name     => 'r2',
    rule_set_name => 'rs');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name     => 'r3',
    rule_set_name => 'rs');
END;
/
/
/*
Query the Data Dictionary
At this point, you can view the evaluation context, rules, and rule set you created in
the previous steps.
*/
COLUMN EVALUATION_CONTEXT_NAME HEADING 'Eval Context Name' FORMAT A30
COLUMN EVALUATION_CONTEXT_COMMENT HEADING 'Eval Context Comment' FORMAT A40
SELECT EVALUATION_CONTEXT_NAME, EVALUATION_CONTEXT_COMMENT
  FROM USER_EVALUATION_CONTEXTS
  ORDER BY EVALUATION_CONTEXT_NAME;

SET LONGCHUNKSIZE 4000
SET LONG 4000
CREATE OR REPLACE PROCEDURE problem_dispatch (priority NUMBER,
            problem_type VARCHAR2 := NULL)
IS
  vvl SYS.RE$VARIABLE_VALUE_LIST;
  truehits SYS.RE$RULE_HIT_LIST;
  maybehits SYS.RE$RULE_HIT_LIST;
  ac SYS.RE$NV_LIST;
  namearray SYS.RE$NAME_ARRAY;
  name VARCHAR2(30);
  cval VARCHAR2(100);
  rnum INTEGER;
  i INTEGER;
  status PLS_INTEGER;
BEGIN
  IF (problem_type IS NULL) THEN
    vvl := SYS.RE$VARIABLE_VALUE_LIST(
      SYS.RE$VARIABLE_VALUE('priority',
                            ANYDATA.CONVERTNUMBER(priority)));
  ELSE
    vvl := SYS.RE$VARIABLE_VALUE_LIST(
      SYS.RE$VARIABLE_VALUE('priority',
                            ANYDATA.CONVERTNUMBER(priority)),
      SYS.RE$VARIABLE_VALUE('problem_type',
                             ANYDATA.CONVERTVARCHAR2(problem_type)));
  END IF;
truehits := SYS.RE$RULE_HIT_LIST();
maybehits := SYS.RE$RULE_HIT_LIST();
DBMS_RULE.EVALUATE(
  rule_set_name => 'support.rs',
  )
  RETURN truehits;
FOR rnum IN 1..truehits.COUNT LOOP
DBMS_OUTPUT.PUT_LINE('Using rule ' || truehits(rnum).rule_name);
ac := truehits(rnum).rule_action_context;
namearray := ac.GET_ALL_NAMES;
FOR i IN 1..namearray.count LOOP
    name := namearray(i);
    status := ac.GET_VALUE(name).GETVARCHAR2(cval);
    IF (name = 'TRUE CENTER') then
        DBMS_OUTPUT.PUT_LINE('Assigning problem to ' || cval);
    ELSIF (name = 'ALERT') THEN
        DBMS_OUTPUT.PUT_LINE('Sending alert to: ' || cval);
    END IF;
END LOOP;
END LOOP;
FOR rnum IN 1..maybehits.COUNT LOOP
DBMS_OUTPUT.PUT_LINE('Using rule ' || maybehits(rnum).rule_name);
ac := maybehits(rnum).rule_action_context;
namearray := ac.GET_ALL_NAMES;
FOR i IN 1..namearray.count LOOP
    name := namearray(i);
    status := ac.GET_VALUE(name).GETVARCHAR2(cval);
    IF (name = 'MAYBE CENTER') then
        DBMS_OUTPUT.PUT_LINE('Assigning problem to ' || cval);
    END IF;
END LOOP;
END LOOP;
END;
/

 Dispatch Sample Problems

The first problem dispatch in this step uses partial evaluation and takes an action based on the partial evaluation. Specifically, the first problem dispatch specifies that the priority is 1 and the problem_type is NULL. In this case, the rules engine returns a MAYBE rule for the event, and the problem_dispatch procedure assigns the problem to the Texas center.

The second and third problem dispatches do not use partial evaluation. Each of these problems evaluate to TRUE for a rule, and the problem is assigned accordingly by the problem_dispatch procedure.

EXECUTE problem_dispatch(1, NULL);
EXECUTE problem_dispatch(2, 'HARDWARE');
EXECUTE problem_dispatch(3, 'SOFTWARE');
/

 Clean Up the Environment (Optional)

You can clean up the sample environment by dropping the support user.

CONNECT SYSTEM
DROP USER support CASCADE;

/*

Check the Spool Results
Check the rules_stored_variables_partial.out spool file to ensure that all actions completed successfully after this script completes.
*/

SET ECHO OFF
SPOOL OFF

/******************** END OF SCRIPT ********************/

6.5 Using Rules on Data Stored in a Table

This example illustrates how to use rules to evaluate data stored in a table. This example is similar to the example described in "Using Rules on Nontable Data Stored in Explicit Variables". In both examples, the application routes customer problems based on priority. However, in this example, the problems are stored in a table instead of variables.

The application uses the problems table in the support schema, into which customer problems are inserted. This example uses the following rules for handling customer problems:

• Assign all problems with priority greater than 2 to the San Jose Center.
• Assign all problems with priority less than or equal to 2 to the New York Center.
• Send an alert to the vice president of support for a problem with priority equal to 1.

The evaluation context consists of the problems table. The relevant row of the table, which corresponds to the problem being routed, is passed to the DBMS_RULE.EVALUATE procedure as a table value.

Complete the following steps:

1. Show Output and Spool Results
2. Create the support User
3. Grant the support User the Necessary System Privileges on Rules
4. Create the problems Table
5. Create the evalctx Evaluation Context
6. Create the Rules that Correspond to Problem Priority
7. Create the rs Rule Set
8. Add the Rules to the Rule Set
9. Create the problem_dispatch PL/SQL Procedure
10. Log Problems
11. Check the Spool Results
/************************* BEGINNING OF SCRIPT *****************************/

Show Output and Spool Results
Run SET ECHO ON and specify the spool file for the script. Check the spool file for errors after you run this script.

*/

SET ECHO ON
SPOOL rules_table.out

/*

Create the support User

*/

CONNECT SYSTEM

CREATE TABLESPACE support_tbs1 DATAFILE 'support_tbs1.dbf'
  SIZE 5M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;

ACCEPT password PROMPT 'Enter password for user: ' HIDE

CREATE USER support
  IDENTIFIED BY &password
  DEFAULT TABLESPACE support_tbs1
  QUOTA UNLIMITED ON support_tbs1;

GRANT ALTER SESSION, CREATE CLUSTER, CREATE DATABASE LINK, CREATE SEQUENCE,
  CREATE SESSION, CREATE SYNONYM, CREATE TABLE, CREATE VIEW, CREATE INDEXTYPE,
  CREATE OPERATOR, CREATE PROCEDURE, CREATE TRIGGER, CREATE TYPE
TO support;

/*

Grant the support User the Necessary System Privileges on Rules

*/

BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege => DBMS_RULE_ADM.CREATE_RULE_SET_OBJ,
    grantee => 'support',
    grant_option => FALSE);
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
privilege => DBMS_RULE_ADM.CREATE_RULE_OBJ,
grantee => 'support',
grant_option => FALSE);
DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
  privilege => DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT_OBJ,
grantee => 'support',
  grant_option => FALSE);
END;
/

_Create the problems Table_
*/

CONNECT support

SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 100
SET SERVEROUTPUT ON

CREATE TABLE problems(
  probid NUMBER PRIMARY KEY,
custid NUMBER,
priority NUMBER,
description VARCHAR2(4000),
center VARCHAR2(100));

/*
_Create the evalctx Evaluation Context_
*/

DECLARE
  ta SYS.RE$TABLE_ALIAS_LIST;
BEGIN
  ta := SYS.RE$TABLE_ALIAS_LIST(SYS.RE$TABLE_ALIAS('prob', 'problems'));
  DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT(
    evaluation_context_name => 'evalctx',
    table_aliases => ta,
    evaluation_context_comment => 'support problem definition');
END;
/

/*
_Create the Rules that Correspond to Problem Priority_
The following code creates one action context for each rule, and one name-value pair in each action context.
*/
DECLARE
   ac  SYS.RE$NV_LIST;
BEGIN
   ac := SYS.RE$NV_LIST(NULL);
   ac.ADD_PAIR('CENTER', ANYDATA.CONVERTVARCHAR2('San Jose'));
   DBMS_RULE_ADM.CREATE_RULE(
      rule_name => 'r1',
      condition => 'prob.priority > 2',
      action_context => ac,
      rule_comment => 'Low priority problems');
   ac := SYS.RE$NV_LIST(NULL);
   ac.ADD_PAIR('CENTER', ANYDATA.CONVERTVARCHAR2('New York'));
   DBMS_RULE_ADM.CREATE_RULE(
      rule_name => 'r2',
      condition => 'prob.priority <= 2',
      action_context => ac,
      rule_comment => 'High priority problems');
   ac := SYS.RE$NV_LIST(NULL);
   ac.ADD_PAIR('ALERT', ANYDATA.CONVERTVARCHAR2('John Doe'));
   DBMS_RULE_ADM.CREATE_RULE(
      rule_name => 'r3',
      condition => 'prob.priority = 1',
      action_context => ac,
      rule_comment => 'Urgent problems');
END;
/
/
/*
Create the rs Rule Set
*/
BEGIN
   DBMS_RULE_ADM.CREATE_RULE_SET(
      rule_set_name => 'rs',
      evaluation_context => 'evalctx',
      rule_set_comment => 'support rules');
END;
/
/
/*
Add the Rules to the Rule Set
*/
BEGIN
   DBMS_RULE_ADM.ADD_RULE(
      rule_name => 'r1',
      rule_set_name => 'rs');
   DBMS_RULE_ADM.ADD_RULE(
      rule_name => 'r2',
      rule_set_name => 'rs');
   DBMS_RULE_ADM.ADD_RULE(
      rule_name => 'r3',
      rule_set_name => 'rs');
END;
CREATE OR REPLACE PROCEDURE problem_dispatch
IS
   cursor c IS SELECT probid, rowid FROM problems WHERE center IS NULL;
   tv SYS.RE$TABLE_VALUE;
   tvl SYS.RE$TABLE_VALUE_LIST;
   truehits SYS.RE$RULE_HIT_LIST;
   maybehits SYS.RE$RULE_HIT_LIST;
   ac SYS.RE$NV_LIST;
   namearray SYS.RE$NAME_ARRAY;
   name VARCHAR2(30);
   cval VARCHAR2(100);
   rnum INTEGER;
   i INTEGER;
   status PLS_INTEGER;
BEGIN
   FOR r IN c LOOP
      tv := SYS.RE$TABLE_VALUE('prob', rowidtochar(r.rowid));
      tvl := SYS.RE$TABLE_VALUE_LIST(tv);
      truehits := SYS.RE$RULE_HIT_LIST();
      maybehits := SYS.RE$RULE_HIT_LIST();
      DBMS_RULE.EVALUATE(
         rule_set_name => 'support.rs',
         evaluation_context => 'evalctx',
         table_values => tvl,
         true_rules => truehits,
         maybe_rules => maybehits);
      FOR rnum IN 1..truehits.COUNT LOOP
         DBMS_OUTPUT.PUT_LINE('Using rule ' || truehits(rnum).rule_name);
         ac := truehits(rnum).rule_action_context;
         namearray := ac.GET_ALL_NAMES;
         FOR i IN 1..namearray.COUNT LOOP
            name := namearray(i);
            status := ac.GET_VALUE(name).GETVARCHAR2(cval);
            IF (name = 'CENTER') THEN
               UPDATE PROBLEMS SET center = cval WHERE rowid = r.rowid;
               DBMS_OUTPUT.PUT_LINE('Assigning ' || r.probid || ' to ' || cval);
            ELSIF (name = 'ALERT') THEN
               DBMS_OUTPUT.PUT_LINE('Alert: ' || 'cval ' || ' Problem:' || r.probid);
            END IF;
         END LOOP;
      END LOOP;
   END LOOP;
END;
/

CREATE the problem_dispatch PL/SQL Procedure

Log Problems

/*
INSERT INTO problems(probid, custid, priority, description)
VALUES(10101, 11, 1, 'no dial tone');

INSERT INTO problems(probid, custid, priority, description)
VALUES(10102, 21, 2, 'noise on local calls');

INSERT INTO problems(probid, custid, priority, description)
VALUES(10103, 31, 3, 'noise on long distance calls');

COMMIT;

/*

Check the Spool Results
Check the rules_table.out spool file to ensure that all actions completed successfully after this script completes.
*/

SET ECHO OFF
SPOOL OFF

/**************************** END OF SCRIPT ******************************/

See Also:

"Dispatching Problems and Checking Results for the Table Examples" for the steps to complete to dispatch the problems logged in this example and check the results of the problem dispatch

6.6 Using Rules on Both Explicit Variables and Table Data

This example illustrates how to use rules to evaluate data stored in explicit variables and in a table. The application uses the problems table in the support schema, into which customer problems are inserted. This example uses the following rules for handling customer problems:

- Assign all problems with priority greater than 2 to the San Jose Center.
- Assign all problems with priority equal to 2 to the New York Center.
- Assign all problems with priority equal to 1 to the Tampa Center from 8 AM to 8 PM.
- Assign all problems with priority equal to 1 to the Bangalore Center from 8 PM to 8 AM.
- Send an alert to the vice president of support for a problem with priority equal to 1.

The evaluation context consists of the problems table. The relevant row of the table, which corresponds to the problem being routed, is passed to the DBMS_RULE.EVALUATE procedure as a table value.

Some of the rules in this example refer to the current time, which is represented as an explicit variable named current_time. The current time is treated as additional data in the evaluation context. It is represented as a variable for the following reasons:
It is not practical to store the current time in a table because it would have to be updated very often.

The current time can be accessed by inserting calls to `SYSDATE` in every rule that requires it, but that would cause repeated invocations of the same SQL function `SYSDATE`, which might slow down rule evaluation. Different values of the current time in different rules might lead to incorrect behavior.

Complete the following steps:

1. Show Output and Spool Results
2. Create the support User
3. Grant the support User the Necessary System Privileges on Rules
4. Create the problems Table
5. Create the evalctx Evaluation Context
6. Create the Rules that Correspond to Problem Priority
7. Create the rs Rule Set
8. Add the Rules to the Rule Set
9. Create the problem_dispatch PL/SQL Procedure
10. Log Problems
11. Check the Spool Results

---

**Note:**

If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line after this note to the next "END OF SCRIPT" line into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to all of the databases in the environment.

```sql
/*BEGINNING OF SCRIPT***************************************************************************/

Show Output and Spool Results
Run `SET ECHO ON` and specify the spool file for the script. Check the spool file for errors after you run this script.

```
```
CREATE TABLESPACE support_tbs2 DATAFILE 'support_tbs2.dbf' 
   SIZE 5M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;

ACCEPT password PROMPT 'Enter password for user: ' HIDE

CREATE USER support
IDENTIFIED BY &password
   DEFAULT TABLESPACE support_tbs2
   QUOTA UNLIMITED ON support_tbs2;

GRANT ALTER SESSION, CREATE CLUSTER, CREATE DATABASE LINK, CREATE SEQUENCE,
CREATE SESSION, CREATE SYNONYM, CREATE TABLE, CREATE VIEW, CREATE INDEXTYPE,
CREATE OPERATOR, CREATE PROCEDURE, CREATE TRIGGER, CREATE TYPE
TO support;

/*

Grant the support User the Necessary System Privileges on Rules
*/

BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege => DBMS_RULE_ADM.CREATE_RULE_SET_OBJ,
    grantee   => 'support',
    grant_option => FALSE);
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege => DBMS_RULE_ADM.CREATE_RULE_OBJ,
    grantee   => 'support',
    grant_option => FALSE);
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege => DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT_OBJ,
    grantee   => 'support',
    grant_option => FALSE);
END;
/

/*

Create the problems Table
*/

CONNECT support

SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
SET TRIMSPOOL ON
SET TAB OFF
SET PAGESIZE 100
SET SERVEROUTPUT ON

CREATE TABLE problems(
  probid          NUMBER PRIMARY KEY,
  custid          NUMBER,
  priority        NUMBER,
)
CREATE the evalctx Evaluation Context

DECLARE
    ta SYS.RE$TABLE_ALIAS_LIST;
    vt SYS.RE$VARIABLE_TYPE_LIST;
BEGIN
    ta := SYS.RE$TABLE_ALIAS_LIST(SYS.RE$TABLE_ALIAS('prob', 'problems'));
    vt := SYS.RE$VARIABLE_TYPE_LIST(
        SYS.RE$VARIABLE_TYPE('current_time', 'DATE', NULL, NULL));
    DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT(
        evaluation_context_name => 'evalctx',
        table_aliases => ta,
        variable_types => vt,
        evaluation_context_comment => 'support problem definition');
END;
/

CREATE the Rules that Correspond to Problem Priority
The following code creates one action context for each rule, and one name-value pair in each action context.

DECLARE
    ac SYS.RE$NV_LIST;
BEGIN
    ac := SYS.RE$NV_LIST(NULL);
    ac.ADD_PAIR('CENTER', ANYDATA.CONVERTVARCHAR2('San Jose'));
    DBMS_RULE_ADM.CREATE_RULE(
        rule_name => 'r1',
        condition => 'prob.priority > 2',
        action_context => ac,
        rule_comment => 'Low priority problems');
    ac := SYS.RE$NV_LIST(NULL);
    ac.ADD_PAIR('CENTER', ANYDATA.CONVERTVARCHAR2('New York'));
    DBMS_RULE_ADM.CREATE_RULE(
        rule_name => 'r2',
        condition => 'prob.priority = 2',
        action_context => ac,
        rule_comment => 'High priority problems');
    ac := SYS.RE$NV_LIST(NULL);
    ac.ADD_PAIR('CENTER', ANYDATA.CONVERTVARCHAR2('Tampa'));
    DBMS_RULE_ADM.CREATE_RULE(
        rule_name => 'r3',
        condition => 'prob.priority = 1',
        action_context => ac,
        rule_comment => 'Urgent problems');
END;
rule_name => 'r4',
condition => '(prob.priority = 1) and ' ||
  '((TO_NUMBER(TO_CHAR(:current_time, ''HH24'')) >= 8) and ' ||
    'TO_NUMBER(TO_CHAR(:current_time, ''HH24'')) <= 20)',
action_context => ac,
rule_comment => 'Urgent daytime problems');
ac := sys.RE$NV_LIST(NULL);
ac.add_pair('CENTER', ANYDATA.CONVERTVARCHAR2('Bangalore'));
DBMS_RULE_ADM.CREATE_RULE(
  rule_name => 'r5',
  condition => '(prob.priority = 1) and ' ||
    '((TO_NUMBER(TO_CHAR(:current_time, ''HH24'')) < 8) or ' ||
      'TO_NUMBER(TO_CHAR(:current_time, ''HH24'')) > 20))',
  action_context => ac,
  rule_comment => 'Urgent nighttime problems');
END;
/
/*
Create the rs Rule Set
*/
BEGIN
 DBMS_RULE_ADM.CREATE_RULE_SET(
   rule_set_name => 'rs',
   evaluation_context => 'evalctx',
   rule_set_comment => 'support rules');
END;
/
/*
Add the Rules to the Rule Set
*/
BEGIN
 DBMS_RULE_ADM.ADD_RULE(
   rule_name => 'r1',
   rule_set_name => 'rs');
 DBMS_RULE_ADM.ADD_RULE(
   rule_name => 'r2',
   rule_set_name => 'rs');
 DBMS_RULE_ADM.ADD_RULE(
   rule_name => 'r3',
   rule_set_name => 'rs');
 DBMS_RULE_ADM.ADD_RULE(
   rule_name => 'r4',
   rule_set_name => 'rs');
 DBMS_RULE_ADM.ADD_RULE(
   rule_name => 'r5',
   rule_set_name => 'rs');
END;
/
/*
Create the problem_dispatch PL/SQL Procedure

CREATE OR REPLACE PROCEDURE problem_dispatch
IS
  cursor c is SELECT probid, rowid FROM PROBLEMS WHERE center IS NULL;
  tv  SYS.RE$TABLE_VALUE;
  tvl SYS.RE$TABLE_VALUE_LIST;
  vv1 SYS.RE$VARIABLE_VALUE;
  vvl SYS.RE$VARIABLE_VALUE_LIST;
  truehits SYS.RE$RULE_HIT_LIST;
  maybehits SYS.RE$RULE_HIT_LIST;
  ac  SYS.RE$NV_LIST;
  namearray SYS.RE$NAME_ARRAY;
  name VARCHAR2(30);
  cval VARCHAR2(100);
  rnum INTEGER;
  i   INTEGER;
  status PLS_INTEGER;
BEGIN
  FOR r IN c LOOP
    tv := SYS.RE$TABLE_VALUE('prob', ROWIDTOCHAR(r.rowid));
    tvl := SYS.RE$TABLE_VALUE_LIST(tv);
    vv1 := SYS.RE$VARIABLE_VALUE('current_time',
      ANYDATA.CONVERTDATE(SYSDATE));
    vvl := SYS.RE$VARIABLE_VALUE_LIST(vv1);
    truehits := SYS.RE$RULE_HIT_LIST();
    maybehits := SYS.RE$RULE_HIT_LIST();
    DBMS_RULE.EVALUATE(
      rule_set_name => 'support.rs',
      evaluation_context => 'evalctx',
      table_values => tvl,
      variable_values => vvl,
      true_rules => truehits,
      maybe_rules => maybehits);
    FOR rnum IN 1..truehits.COUNT loop
      DBMS_OUTPUT.PUT_LINE('Using rule: ' || truehits(rnum).rule_name);
      ac := truehits(rnum).rule_action_context;
      namearray := ac.GET_ALL_NAMES;
      FOR i in 1..namearray.COUNT LOOP
        name := namearray(i);
        status := ac.GET_VALUE(name).GETVARCHAR2(cval);
        IF (name = 'CENTER') THEN
          UPDATE problems SET center = cval
          WHERE rowid = r.rowid;
          DBMS_OUTPUT.PUT_LINE('Assigning: ' || r.probid || ' to ' || cval);
        ELSIF (name = 'ALERT') THEN
          DBMS_OUTPUT.PUT_LINE('Alert: ' || cval || ' Problem: ' || r.probid);
        END IF;
      END LOOP;
    END LOOP;
  END LOOP;
END;
/
Log Problems

*/

INSERT INTO problems(probid, custid, priority, description)
VALUES(10201, 12, 1, 'no dial tone');

INSERT INTO problems(probid, custid, priority, description)
VALUES(10202, 22, 2, 'noise on local calls');

INSERT INTO problems(probid, custid, priority, description)
VALUES(10203, 32, 3, 'noise on long distance calls');

COMMIT;

/*

Check the Spool Results

Check the rules_var_tab.out spool file to ensure that all actions completed successfully after this script completes.

*/

SET ECHO OFF
SPOOL OFF

/*************************** END OF SCRIPT ******************************/

See Also:

"Dispatching Problems and Checking Results for the Table Examples" for the steps to complete to dispatch the problems logged in this example and check the results of the problem dispatch

6.7 Using Rules on Implicit Variables and Table Data

This example illustrates how to use rules to evaluate implicit variables and data stored in a table. The application uses the problems table in the support schema, into which customer problems are inserted. This example uses the following rules for handling customer problems:

• Assign all problems with priority greater than 2 to the San Jose Center.
• Assign all problems with priority equal to 2 to the New York Center.
• Assign all problems with priority equal to 1 to the Tampa Center from 8 AM to 8 PM.
• Assign all problems with priority equal to 1 to the Bangalore Center after 8 PM and before 8 AM.
• Send an alert to the vice president of support for a problem with priority equal to 1.

The evaluation context consists of the problems table. The relevant row of the table, which corresponds to the problem being routed, is passed to the DBMS_RULE.EVALUATE procedure as a table value.
As in the example illustrated in "Using Rules on Both Explicit Variables and Table Data", the current time is represented as a variable named `current_time`. However, this variable value is not specified during evaluation by the caller. That is, `current_time` is an implicit variable in this example. A PL/SQL function named `timefunc` is specified for `current_time`, and this function is invoked once during evaluation to get its value.

Using implicit variables can be useful in other cases if one of the following conditions is true:

- The caller does not have access to the variable value.
- The variable is referenced infrequently in rules. Because it is implicit, its value can be retrieved only when necessary, and does not need to be passed in for every evaluation.

Complete the following steps:

1. Show Output and Spool Results
2. Create the support User
3. Grant the support User the Necessary System Privileges on Rules
4. Create the problems Table
5. Create the timefunc Function to Return the Value of `current_time`
6. Create the evalctx Evaluation Context
7. Create the Rules that Correspond to Problem Priority
8. Create the rs Rule Set
9. Add the Rules to the Rule Set
10. Create the problem_dispatch PL/SQL Procedure
11. Log Problems
12. Check the Spool Results

---

**Note:**

If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line after this note to the next "END OF SCRIPT" line into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to all of the databases in the environment.

/******************** BEGINNING OF SCRIPT ******************/

**Show Output and Spool Results**

Run `SET ECHO ON` and specify the spool file for the script. Check the spool file for errors after you run this script.

*/

SET ECHO ON
SPOOL rules_implicit_var.out
CREATE TABLESPACE support_tbs3 DATAFILE 'support_tbs3.dbf'
  SIZE 5M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;

ACCEPT password PROMPT 'Enter password for user: ' HIDE

CREATE USER support
  IDENTIFIED BY &password
  DEFAULT TABLESPACE support_tbs3
  QUOTA UNLIMITED ON support_tbs3;

GRANT ALTER SESSION, CREATE CLUSTER, CREATE DATABASE LINK, CREATE SEQUENCE,
  CREATE SESSION, CREATE SYNONYM, CREATE TABLE, CREATE VIEW, CREATE INDEXTYPE,
  CREATE OPERATOR, CREATE PROCEDURE, CREATE TRIGGER, CREATE TYPE
  TO support;

BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege => DBMS_RULE_ADM.CREATE_RULE_SET_OBJ,
    grantee => 'support',
    grant_option => FALSE);
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege => DBMS_RULE_ADM.CREATE_RULE_OBJ,
    grantee => 'support',
    grant_option => FALSE);
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege => DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT_OBJ,
    grantee => 'support',
    grant_option => FALSE);
END;
/

CREATE USER support

CONNECT support

SET FEEDBACK 1
SET NUMWIDTH 10
SET LINESIZE 80
CREATE TABLE problems(
    probid          NUMBER PRIMARY KEY,
    custid          NUMBER,
    priority        NUMBER,
    description     VARCHAR2(4000),
    center          VARCHAR2(100));
/

Create the timefunc Function to Return the Value of current_time
/

CREATE OR REPLACE FUNCTION timefunc(
    eco    VARCHAR2,
    ecn    VARCHAR2,
    var    VARCHAR2,
    evctx  SYS.RE$NV_LIST)
RETURN SYS.RE$VARIABLE_VALUE
IS
BEGIN
    IF (var = 'CURRENT_TIME') THEN
        RETURN(SYS.RE$VARIABLE_VALUE('current_time',
                                        ANYDATA.CONVERTDATE(SYSDATE)));
    ELSE
        RETURN(NULL);
    END IF;
END;
/

Create the evalctx Evaluation Context
/

DECLARE
    ta SYS.RE$TABLE_ALIAS_LIST;
    vt SYS.RE$VARIABLE_TYPE_LIST;
BEGIN
    ta := SYS.RE$TABLE_ALIAS_LIST(SYS.RE$TABLE_ALIAS('prob', 'problems'));
    vt := SYS.RE$VARIABLE_TYPE_LIST(
        SYS.RE$VARIABLE_TYPE('current_time', 'DATE', 'timefunc', NULL));
    DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT(
        evaluation_context_name    => 'evalctx',
        table_aliases              => ta,
        variable_types             => vt,
        evaluation_context_comment => 'support problem definition');
END;
/
/*
Create the Rules that Correspond to Problem Priority

The following code creates one action context for each rule, and one name-value pair in each action context.

```sql
DECLARE
  ac SYS.RE$NV_LIST;
BEGIN
  ac := SYS.RE$NV_LIST(NULL);
  ac.ADD_PAIR('CENTER', ANYDATA.CONVERTVARCHAR2('San Jose'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name => 'r1',
    condition => 'prob.priority > 2',
    action_context => ac,
    rule_comment => 'Low priority problems');
  ac := SYS.RE$NV_LIST(NULL);
  ac.ADD_PAIR('CENTER', ANYDATA.CONVERTVARCHAR2('New York'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name => 'r2',
    condition => 'prob.priority = 2',
    action_context => ac,
    rule_comment => 'High priority problems');
  ac := SYS.RE$NV_LIST(NULL);
  ac.ADD_PAIR('ALERT', ANYDATA.CONVERTVARCHAR2('John Doe'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name => 'r3',
    condition => 'prob.priority = 1',
    action_context => ac,
    rule_comment => 'Urgent problems');
  ac := SYS.RE$NV_LIST(NULL);
  ac.ADD_PAIR('CENTER', ANYDATA.CONVERTVARCHAR2('Tampa'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name => 'r4',
    condition => '(prob.priority = 1) and ' ||
      '((TO_NUMBER(TO_CHAR(:current_time, 'HH24')) >= 8) and ' ||
      '((TO_NUMBER(TO_CHAR(:current_time, 'HH24')) <= 20))',
    action_context => ac,
    rule_comment => 'Urgent daytime problems');
  ac := SYS.RE$NV_LIST(NULL);
  ac.ADD_PAIR('CENTER', ANYDATA.CONVERTVARCHAR2('Bangalore'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name => 'r5',
    condition => '(prob.priority = 1) and ' ||
      '((TO_NUMBER(TO_CHAR(:current_time, 'HH24')) < 8) or ' ||
      '((TO_NUMBER(TO_CHAR(:current_time, 'HH24')) > 20))',
    action_context => ac,
    rule_comment => 'Urgent nighttime problems');
END;
/
/
/
CREATE the rs Rule Set
/
/
BEGIN
  DBMS_RULE_ADM.CREATE_RULE_SET(
```
rule_set_name => 'rs',
evaluation_context => 'evalctx',
rule_set_comment => 'support rules');
END;
/

Add the Rules to the Rule Set
/

BEGIN
  DBMS_RULE_ADM.ADD_RULE(
    rule_name => 'r1',
    rule_set_name => 'rs');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name => 'r2',
    rule_set_name => 'rs');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name => 'r3',
    rule_set_name => 'rs');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name => 'r4',
    rule_set_name => 'rs');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name => 'r5',
    rule_set_name => 'rs');
END;
/
/*
Create the problem_dispatch PL/SQL Procedure
*/

CREATE OR REPLACE PROCEDURE problem_dispatch
IS
  cursor c IS SELECT probid, rowid FROM problems WHERE center IS NULL;
tv SYS.RE$TABLE_VALUE;
tvl SYS.RE$TABLE_VALUE_LIST;
truehits SYS.RE$RULE_HIT_LIST;
maybehits SYS.RE$RULE_HIT_LIST;
ac SYS.RE$NV_LIST;
namearray SYS.RE$NAME_ARRAY;
name VARCHAR2(30);
cval VARCHAR2(100);
rnum INTEGER;
i INTEGER;
status PLS_INTEGER;
BEGIN
  FOR r IN c LOOP
    tv := SYS.RE$TABLE_VALUE('prob', rowidtochar(r.rowid));
tvl := SYS.RE$TABLE_VALUE_LIST(tv);
    truehits := SYS.RE$RULE_HIT_LIST();
    maybehits := SYS.RE$RULE_HIT_LIST();
    DBMS_RULE.EVALUATE(
rule_set_name   => 'support.rs',
evaluation_context => 'evalctx',
table_values    => tvl,
true_rules      => truehits,
maybe_rules     => maybehits);

FOR rnum IN 1..truehits.COUNT LOOP
  DBMS_OUTPUT.PUT_LINE('Using rule ' || truehits(rnum).rule_name);
  ac := truehits(rnum).rule_action_context;
  namearray := ac.GET_ALL_NAMES;
  FOR i IN 1..namearray.COUNT LOOP
    name := namearray(i);
    status := ac.GET_VALUE(name).GETVARCHAR2(cval);
    IF (name = 'CENTER') THEN
      UPDATE problems SET center = cval
      WHERE rowid = r.rowid;
      DBMS_OUTPUT.PUT_LINE('Assigning ' || r.probid || ' to ' || cval);
    ELSIF (name = 'ALERT') THEN
      DBMS_OUTPUT.PUT_LINE('Alert: ' || cval || ' Problem: ' || r.probid);
    END IF;
  END LOOP;
END LOOP;
END;
/

Log Problems
/*

INSERT INTO problems(probid, custid, priority, description)
  VALUES(10301, 13, 1, 'no dial tone');

INSERT INTO problems(probid, custid, priority, description)
  VALUES(10302, 23, 2, 'noise on local calls');

INSERT INTO problems(probid, custid, priority, description)
  VALUES(10303, 33, 3, 'noise on long distance calls');

COMMIT;
/

Check the Spool Results
Check the rules_implicit_var.out spool file to ensure that all actions completed successfully after this script completes.
/*

SET ECHO OFF
SPOOL OFF

/**************************** END OF SCRIPT ****************************/
6.8 Using Event Contexts and Implicit Variables with Rules

An event context is a varray of type `SYS.RE$NV_LIST` that contains name-value pairs that contain information about the event. This optional information is not directly used or interpreted by the rules engine. Instead, it is passed to client callbacks such as an evaluation function, a variable value function (for implicit variables), or a variable method function.

In this example, assume every customer has a primary contact person, and the goal is to assign the problem reported by a customer to the support center to which the customer's primary contact person belongs. The customer name is passed in the event context.

This example illustrates how to use event contexts with rules to evaluate implicit variables. Specifically, when an event is evaluated using the `DBMS_RULE.EVALUATE` procedure, the event context is passed to the variable value function for implicit variables in the evaluation context. The name of the variable value function is `find_contact`, and this PL/SQL function returns the contact person based on the name of the company specified in the event context. The rule set is evaluated based on the contact person name and the priority for an event.

This example uses the following rules for handling customer problems:

- Assign all problems that belong to Jane to the San Jose Center.
- Assign all problems that belong to Fred to the New York Center.
- Assign all problems whose primary contact is unknown to George at the Texas Center.
- Send an alert to the vice president of support for a problem with priority equal to 1.

Complete the following steps:

1. Show Output and Spool Results
2. Create the support User
3. Grant the support User the Necessary System Privileges on Rules
4. Create the find_contact Function to Return a Customer's Contact
5. Create the evalctx Evaluation Context
6. Create the Rules that Correspond to Problem Priority and Contact
7. Create the rs Rule Set
8. Add the Rules to the Rule Set
9. Query the Data Dictionary
10. Create the problem_dispatch PL/SQL Procedure
11. Dispatch Sample Problems
12. Clean Up the Environment (Optional)

13. Check the Spool Results

Note:
If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line after this note to the next "END OF SCRIPT" line into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to all of the databases in the environment.

/** ****************************************** BEGINNING OF SCRIPT *********************************************/

Show Output and Spool Results
Run SET ECHO ON and specify the spool file for the script. Check the spool file for errors after you run this script.

/*

SET ECHO ON
SPOOL rules_event_context.out

*/

Create the support User

/*

CONNECT SYSTEM

ACCEPT password PROMPT 'Enter password for user: ' HIDE

GRANT ALTER SESSION, CREATE CLUSTER, CREATE DATABASE LINK, CREATE SEQUENCE, CREATE SESSION, CREATE SYNONYM, CREATE TABLE, CREATE VIEW, CREATE INDEXTYPE, CREATE OPERATOR, CREATE PROCEDURE, CREATE TRIGGER, CREATE TYPE TO support IDENTIFIED BY &support;

*/

Grant the support User the Necessary System Privileges on Rules

/*

BEGIN

DBMS_RULEADM.GRANT_SYSTEM_PRIVILEGE(
    privilege => DBMS_RULEADM.CREATE_RULE_SET_OBJ,
    grantee => 'support',
    grant_option => FALSE);

DBMS_RULEADM.GRANT_SYSTEM_PRIVILEGE(
    privilege => DBMS_RULEADM.CREATE_RULE_OBJ,
    grantee => 'support',
    grant_option => FALSE);

DBMS_RULEADM.GRANT_SYSTEM_PRIVILEGE(
 */
privilege => DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT_OBJ,
grantee => 'support',
grant_option => FALSE);
END;
/

Create the find_contact Function to Return a Customer's Contact

CREATE OR REPLACE FUNCTION find_contact(
eco       VARCHAR2,
ecn       VARCHAR2,
var       VARCHAR2,
evctx     SYS.RE$NV_LIST)
RETURN SYS.RE$VARIABLE_VALUE IS
    cust      VARCHAR2(30);
    contact   VARCHAR2(30);
    status    PLS_INTEGER;
BEGIN
    IF (var = 'CUSTOMER_CONTACT') THEN
        status := evctx.GET_VALUE('CUSTOMER').GETVARCHAR2(cust);
        IF (cust = 'COMPANY1') THEN     -- COMPANY1's contact person is Jane
            contact := 'JANE';
        ELSEIF (cust = 'COMPANY2') THEN  -- COMPANY2's contact person is Fred
            contact := 'FRED';
        ELSE        -- Assign customers without primary contact person to George
            contact := 'GEORGE';
        END IF;
        RETURN SYS.RE$VARIABLE_VALUE('customer_contact',
                                      ANYDATA.CONVERTVARCHAR2(contact));
    ELSE
        RETURN NULL;
    END IF;
END;
/

Create the evalctx Evaluation Context

DECLARE
    vt SYS.RE$VARIABLE_TYPE_LIST;
BEGIN
    vt := SYS.RE$VARIABLE_TYPE_LIST(
SYS.RE$VARIABLE_TYPE('priority', 'NUMBER', NULL, NULL),
SYS.RE$VARIABLE_TYPE('customer_contact', 'VARCHAR2(30)',
   'find_contact', NULL));

DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT(
    evaluation_context_name    => 'evalctx',
    variable_types             => vt,
    evaluation_context_comment => 'support problem definition');

END;
/

create the rules that correspond to problem priority and contact
the following code creates one action context for each rule, and one name-value pair
in each action context.
*/

DECLARE
  ac  SYS.RE$NV_LIST;
BEGIN
  ac := SYS.RE$NV_LIST(NULL);
  ac.ADD_PAIR('CENTER', ANYDATA.CONVERTVARCHAR2('San Jose'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name       => 'r1',
    condition       => ':customer_contact = 'JANE'' ,
    action_context  => ac,
    rule_comment    => 'Jane''s customer problems');
  ac := sys.re$nv_list(NULL);
  ac.ADD_PAIR('CENTER', ANYDATA.CONVERTVARCHAR2('New York'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name       => 'r2',
    condition       => ':customer_contact = 'FRED'' ,
    action_context  => ac,
    rule_comment    => 'Fred''s customer problems');
  ac := sys.re$nv_list(NULL);
  ac.ADD_PAIR('CENTER', ANYDATA.CONVERTVARCHAR2('Texas'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name       => 'r3',
    condition       => ':customer_contact = 'GEORGE'' ,
    action_context  => ac,
    rule_comment    => 'George''s customer problems');
  ac := sys.re$nv_list(NULL);
  ac.ADD_PAIR('ALERT', ANYDATA.CONVERTVARCHAR2('John Doe'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name       => 'r4',
    condition       => ':priority=1',
    action_context  => ac,
    rule_comment    => 'Urgent problems');

END;
/

create the rs rule set
*/

BEGIN
DBMS_RULE_ADM.CREATE_RULE_SET(
    rule_set_name => 'rs',
    evaluation_context => 'evalctx',
    rule_set_comment => 'support rules');
END;
/
/*

Add the Rules to the Rule Set
*/

BEGIN
    DBMS_RULE_ADM.ADD_RULE(
        rule_name => 'r1',
        rule_set_name => 'rs');
    DBMS_RULE_ADM.ADD_RULE(
        rule_name => 'r2',
        rule_set_name => 'rs');
    DBMS_RULE_ADM.ADD_RULE(
        rule_name => 'r3',
        rule_set_name => 'rs');
    DBMS_RULE_ADM.ADD_RULE(
        rule_name => 'r4',
        rule_set_name => 'rs');
END;
/
/*

Query the Data Dictionary
At this point, you can view the evaluation context, rules, and rule set you created in
the previous steps.
*/

COLUMN EVALUATION_CONTEXT_NAME HEADING 'Eval Context Name' FORMAT A30
COLUMN EVALUATION_CONTEXT_COMMENT HEADING 'Eval Context Comment' FORMAT A40

SELECT EVALUATION_CONTEXT_NAME, EVALUATION_CONTEXT_COMMENT
    FROM USER_EVALUATION_CONTEXTS
    ORDER BY EVALUATION_CONTEXT_NAME;

SET LONGCHUNKSIZE 4000
SET LONG 4000
COLUMN RULE_NAME HEADING 'Rule Name' FORMAT A5
COLUMN RULE_CONDITION HEADING 'Rule Condition' FORMAT A35
COLUMN ACTION_CONTEXT_NAME HEADING 'Action Context Name' FORMAT A10
COLUMN ACTION_CONTEXT_VALUE HEADING 'Action Context Value' FORMAT A10

SELECT RULE_NAME,
    RULE_CONDITION,
    AC.NVN_NAME ACTION_CONTEXT_NAME,
    AC.NVN_VALUE.ACCESSVARCHAR2() ACTION_CONTEXT_VALUE
    FROM USER_RULES R, TABLE(R.RULE_ACTION_CONTEXT.ACTX_LIST) AC
    ORDER BY RULE_NAME;

COLUMN RULE_SET_NAME HEADING 'Rule Set Name' FORMAT A20
CREATE OR REPLACE PROCEDURE problem_dispatch (priority NUMBER, customer VARCHAR2)
IS
  vvl SYS.RE$VARIABLE_VALUE_LIST;
  truehits SYS.RE$RULE_HIT_LIST;
  maybehits SYS.RE$RULE_HIT_LIST;
  ac SYS.RE$NV_LIST;
  namearray SYS.RE$NAME_ARRAY;
  name VARCHAR2(30);
  cval VARCHAR2(100);
  rnum INTEGER;
  i INTEGER;
  status PLS_INTEGER;
  evctx SYS.RE$NV_LIST;
BEGIN
  vvl := SYS.RE$VARIABLE_VALUE_LIST(
    SYS.RE$VARIABLE_VALUE('priority',
      ANYDATA.CONVERTNUMBER(priority)));
  evctx := SYS.RE$NV_LIST(NULL);
  evctx.ADD_PAIR('CUSTOMER', ANYDATA.CONVERTVARCHAR2(customer));
  truehits := SYS.RE$RULE_HIT_LIST();
  maybehits := SYS.RE$RULE_HIT_LIST();
  DBMS_RULE.EVALUATE(
    rule_set_name => 'support.rs',
    evaluation_context => 'evalctx',
    event_context => evctx,
    variable_values => vvl,
    true_rules => truehits,
    maybe_rules => maybehits);
  FOR rnum IN 1..truehits.COUNT LOOP
    DBMS_OUTPUT.PUT_LINE('Using rule ' || truehits(rnum).rule_name);
    ac := truehits(rnum).rule_action_context;
    namearray := ac.GET_ALL_NAMES;
    FOR i IN 1..namearray.count LOOP
      name := namearray(i);
      status := ac.GET_VALUE(name).GETVARCHAR2(cval);
      IF (name = 'CENTER') THEN
        DBMS_OUTPUT.PUT_LINE('Assigning problem to ' || cval);
      ELSIF (name = 'ALERT') THEN
        DBMS_OUTPUT.PUT_LINE('Sending alert to: ' || cval);
      END IF;
    END LOOP;
  END LOOP;
END;
Dispatch Sample Problems
The first problem dispatch in this step uses the event context and the variable value function to determine the contact person for COMPANY1. The event context is passed to the find_contact variable value function, and this function returns the contact name JANE. Therefore, rule r1 evaluates to TRUE. The problem_dispatch procedure sends the problem to the San Jose office because JANE belongs to that office. In addition, the priority for this event is 1, which causes rule r4 to evaluate to TRUE. As a result, the problem_dispatch procedure sends an alert to John Doe.

The second problem dispatch in this step uses the event context and the variable value function to determine the contact person for COMPANY2. The event context is passed to the find_contact variable value function, and this function returns the contact name FRED. Therefore, rule r2 evaluates to TRUE. The problem_dispatch procedure sends the problem to the New York office because FRED belongs to that office.

The third problem dispatch in this step uses the event context and the variable value function to determine the contact person for COMPANY3. This company does not have a dedicated contact person. The event context is passed to the find_contact variable value function, and this function returns the contact name GEORGE, because GEORGE is the default contact when no contact person is found. Therefore, rule r3 evaluates to TRUE. The problem_dispatch procedure sends the problem to the Texas office because GEORGE belongs to that office.

Clean Up the Environment (Optional)
You can clean up the sample environment by dropping the support user.

Check the Spool Results
Check the rules_event_context.out spool file to ensure that all actions completed successfully after this script completes.
6.9 Dispatching Problems and Checking Results for the Table Examples

The following sections configure a `problem_dispatch` procedure that updates information in the `problems` table:

- "Using Rules on Data Stored in a Table"
- "Using Rules on Both Explicit Variables and Table Data"
- "Using Rules on Implicit Variables and Table Data"

Complete the following steps to dispatch the problems by running the `problem_dispatch` procedure and display the results in the `problems` table:

1. Query the Data Dictionary
2. List the Problems in the `problems` Table
3. Dispatch the Problems by Running the `problem_dispatch` Procedure
4. List the Problems in the `problems` Table
5. Clean Up the Environment (Optional)

**Query the Data Dictionary**

View the evaluation context, rules, and rule set you created in the example:

```sql
CONNECT support
Enter password: password

COLUMN EVALUATION_CONTEXT_NAME HEADING 'Eval Context Name' FORMAT A30
COLUMN EVALUATION_CONTEXT_COMMENT HEADING 'Eval Context Comment' FORMAT A40

SELECT EVALUATION_CONTEXT_NAME, EVALUATION_CONTEXT_COMMENT
FROM USER_EVALUATION_CONTEXTS
ORDER BY EVALUATION_CONTEXT_NAME;

SET LONGCHUNKSIZE 4000
SET LONG 4000
COLUMN RULE_NAME HEADING 'Rule|Name' FORMAT A5
COLUMN RULE_CONDITION HEADING 'Rule Condition' FORMAT A35
COLUMN ACTION_CONTEXT_NAME HEADING 'Action|Context|Name' FORMAT A10
COLUMN ACTION_CONTEXT_VALUE HEADING 'Action|Context|Value' FORMAT A10

SELECT RULE_NAME,
    RULE_CONDITION,
    AC.NVN_NAME ACTION_CONTEXT_NAME,
    AC.NVN_VALUE.ACCESSVARCHAR2() ACTION_CONTEXT_VALUE
FROM USER_RULES R, TABLE(R.RULE_ACTION_CONTEXT.ACTX_LIST) AC
ORDER BY RULE_NAME;

COLUMN RULE_SET_NAME HEADING 'Rule Set Name' FORMAT A20
COLUMN RULE_SET_EVAL_CONTEXT_OWNER HEADING 'Eval Context|Owner' FORMAT A12
COLUMN RULE_SET_EVAL_CONTEXT_NAME HEADING 'Eval Context Name' FORMAT A25
COLUMN RULE_SET_COMMENT HEADING 'Rule Set|Comment' FORMAT A15

SELECT RULE_SET_NAME,
```
List the Problems in the problems Table
This SELECT statement should show the problems logged previously.

```
SELECT probid, custid, priority, description, center FROM problems
ORDER BY probid;
```

Your output looks similar to the following:

<table>
<thead>
<tr>
<th>Problem ID</th>
<th>Customer ID</th>
<th>Priority</th>
<th>Problem Description</th>
<th>Center</th>
</tr>
</thead>
<tbody>
<tr>
<td>10301</td>
<td>13</td>
<td>1</td>
<td>no dial tone</td>
<td></td>
</tr>
<tr>
<td>10302</td>
<td>23</td>
<td>2</td>
<td>noise on local calls</td>
<td></td>
</tr>
<tr>
<td>10303</td>
<td>33</td>
<td>3</td>
<td>noise on long distance calls</td>
<td></td>
</tr>
</tbody>
</table>

Notice that the Center column is NULL for each new row inserted.

Dispatch the Problems by Running the problem_dispatch Procedure
Execute the `problem_dispatch` procedure.

```
SET SERVEROUTPUT ON
EXECUTE problem_dispatch;
```

List the Problems in the problems Table
If the problems were dispatched successfully in Step Dispatch the Problems by Running the problem_dispatch Procedure, then this SELECT statement should show the center to which each problem was dispatched in the Center column.

```
SELECT probid, custid, priority, description, center FROM problems
ORDER BY probid;
```

Your output looks similar to the following:

<table>
<thead>
<tr>
<th>Problem ID</th>
<th>Customer ID</th>
<th>Priority</th>
<th>Problem Description</th>
<th>Center</th>
</tr>
</thead>
<tbody>
<tr>
<td>10201</td>
<td>12</td>
<td>1</td>
<td>no dial tone</td>
<td>Tampa</td>
</tr>
<tr>
<td>10202</td>
<td>22</td>
<td>2</td>
<td>noise on local calls</td>
<td>New York</td>
</tr>
<tr>
<td>10203</td>
<td>32</td>
<td>3</td>
<td>noise on long distance calls</td>
<td>San Jose</td>
</tr>
</tbody>
</table>

Note: The output will vary depending on which example you used to create the problem_dispatch procedure.

Clean Up the Environment (Optional)
You can clean up the sample environment by dropping the support user.
CONNECT SYSTEM
Enter password: password

DROP USER support CASCADE;
Index

A
ADD SUPPLEMENTAL LOG DATA clause of ALTER TABLE, 3-12
ADD_COLUMN member procedure, 4-5
ADD_PAIR member procedure, 6-2, 6-10, 6-16, 6-27
ALTER TABLE statement
   ADD SUPPLEMENTAL LOG DATA clause, 3-12
apply process
   DML handlers creating, 4-5
   heterogeneous environments example, 2-10
   reenqueue captured LCRs, 4-5
ARCHIVELOG mode
capture process, 1-2, 2-4, 4-2

C
capture process
   ARCHIVELOG mode, 1-2, 2-4, 4-2
CLOSE_ITERATOR procedure, 6-7
COMPATIBLE initialization parameter, 1-2, 2-3
conflict resolution
   MAXIMUM handler example, 3-12
time-based
   example, 3-12
   preparing for, 3-5
constructing
   LCRs, 5-1

D
DBMS_RULE package, 6-1
DBMS_RULE_ADM package, 6-1
DEQUEUE procedure
   example, 4-5
DML handlers, 4-5

E
EVALUATE procedure, 6-2, 6-7, 6-10, 6-16, 6-21, 6-27, 6-34
event contexts
   example, 6-34
EXECUTE member procedure, 4-5

G
GET_ALL_NAMES member function, 6-2, 6-7, 6-10, 6-16, 6-21, 6-27, 6-34
GET_COMMAND_TYPE member function, 4-5
GET_NEXT_HIT function, 6-7
GET_OBJECT_NAME member function, 2-10
GET_VALUE member function rules, 6-2, 6-7, 6-10, 6-16, 6-21, 6-27, 6-34
GET_VALUES member function, 4-5
GLOBAL_NAMES initialization parameter, 1-2, 2-3

I
instantiation
   example, 2-10, 2-23, 3-12

L
LOBs
   logical change records containing, 5-1
   Oracle Streams constructing, 5-1
   logical change records
      LOBs, 5-1
   logical change records (LCRs)
      LOB columns, 5-1

M
messaging client
   example, 4-5
n-way replication
  example, 3-1

Oracle Streams
  adding databases, 2-48
  adding objects, 2-38
  initialization parameters, 1-2, 2-3
  sample environments
    LOBs, 5-1
    replication, 1-1, 2-1, 3-1
    single database, 4-1

propagations
  database links
    creating, 1-3

queues
  ANYDATA
    dequeuing, 4-5

RE$NAME_ARRAY type, 6-16, 6-21, 6-27, 6-34
RE$NV_LIST type, 6-2, 6-10, 6-16, 6-21, 6-27, 6-34
RE$RULE_HIT_LIST type, 6-2, 6-10, 6-16, 6-21, 6-27, 6-34
RE$TABLE_ALIAS_LIST type, 6-16, 6-21, 6-27, 6-34
RE$TABLE_VALUE type, 6-16, 6-21, 6-27, 6-34
RE$TABLE_VALUE_LIST type, 6-16, 6-21, 6-27, 6-34
RE$VARIABLE_TYPE_LIST type, 6-2, 6-10, 6-21, 6-27, 6-34
RE$VARIABLE_VALUE type, 6-2, 6-7, 6-10, 6-21
RE$VARIABLE_VALUE_LIST type, 6-2, 6-7, 6-10, 6-21

reenqueue
  captured LCRs, 4-1

replication
  adding databases, 2-48
  adding objects, 2-38
  heterogeneous single source example, 2-1
  multiple-source example, 3-1
  n-way example, 3-1
  simple single source example, 1-1

rules
  event context
    example, 6-34
  example applications, 6-1
  explicit variables
    example, 6-2, 6-21
  implicit variables
    example, 6-27
  iterative results
    example, 6-7
  MAYBE rules
    example, 6-10
  partial evaluation
    example, 6-10
  table data
    example, 6-16, 6-21, 6-27

SET_COMMAND_TYPE member procedure, 4-5
SET_ENQUEUE_DESTINATION procedure, 4-5
SET_OBJECT_NAME member procedure, 2-10, 4-5
SET_RULE_TRANSFORM_FUNCTION procedure, 2-19
SET_VALUES member procedure, 4-5
supplemental logging
  example, 3-12

transformations
  rule-based
    creating, 2-10
    example, 2-19