Oracle® Database

Using Oracle GoldenGate Classic Architecture with Oracle Database

21c (21.3.0)
F41981-06
July 2023

Oracle Database Using Oracle GoldenGate Classic Architecture with Oracle Database, 21c (21.3.0)

F41981-06

Copyright © 2017, 2023, Oracle and/or its affiliates.

Primary Author: Oracle Corp.

Contents

Preface

The *Step by Step Data Replication Using Oracle GoldenGate Microservices Architecture* is a walk through the entire Oracle GoldenGate data replication cycle using Microservices.

- Audience
- Documentation Accessibility
- Related Information
- Conventions

Audience

This guide is intended for administrators and users who are familiar with Oracle GoldenGate concepts and architecture and who are interested in learning to use the microservices and REST commands for performing various Oracle GoldenGate data replication tasks.

Documentation Accessibility

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

Accessible Access to Oracle Support

Oracle customers who 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` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Related Information

The Oracle GoldenGate Product Documentation Libraries are found at:

Oracle GoldenGate Documentation

Oracle GoldenGate for Big Data Documentation:

https://docs.oracle.com/en/middleware/goldengate/big-data/index.html

For additional information on Oracle GoldenGate, refer to:

https://www.oracle.com/middleware/technologies/goldengate.html

Oracle Database High Availability

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, such as "From the File menu, select **Save**." Boldface also is used for terms defined in text or in the glossary.
italic *`italic`*	Italic type indicates placeholder variables for which you supply particular values, such as in the parameter statement: `TABLE` *`table_name`*. Italic type also is used for book titles and emphasis.

`monospace`
 `MONOSPACE`
	Monospace type indicates code components such as user exits and scripts; the names of files and database objects; URL paths; and input and output text that appears on the screen. Uppercase monospace type is generally used to represent the names of Oracle GoldenGate parameters, commands, and user-configurable functions, as well as SQL commands and keywords.

UPPERCASE	Uppercase in the regular text font indicates the name of a process or utility unless the name is intended to be a specific case. Keywords in upper case (`ADD EXTRACT`, `ADD EXTTRAIL`, `FORMAT RELEASE`).
LOWERCASE	Names of processes to be written in lower case. Examples: `ADD EXTRACT exte`, `ADD EXTRAIL ea`.
{ }	Braces within syntax enclose a set of options that are separated by pipe symbols, one of which must be selected, for example: `{`*`option1`*` | `*`option2`*` | `*`option3`*`}`.
[ ]	Brackets within syntax indicate an optional element. For example in this syntax, the `SAVE` clause is optional: `CLEANUP REPLICAT` *`group_name`* `[, SAVE` *`count`*`]`. Multiple options within an optional element are separated by a pipe symbol, for example: `[`*`option1`*` | `*`option2`*`]`.
Sample Locations	Compass directions such as east, west, north, south to be used for demonstrating Extract and Replicat locations. Datacenters names to use the standard similar to `dc1`, `dc2`.
Group names	Prefixes for each process, as follows: • Extract: ext. Usage with location: `extn`, where ***n*** indicates 'north' compass direction. • Replicat: rep. Usage with location: `repn`, where ***n*** indicates 'north' compass direction. • Distribution Path: `dp`. Usage with location: `dpn`, where ***n*** indicates 'north' compass direction. • Checkpoint table: `ggs_checkpointtable` • Trail file names: e or d depending on whether the trail file is for the Extract of distribution path. Suffix derived in alphabetical order. Usage for an Extract trail file: `ea`, `eb`, `ec`. • Trail file subdirectory: The name will use compass directions to refer to the trail subdirectories. Example for trail subdirectory name would be `/east`, `/west`, `/north`, `/south`.

1

Preparing the Database for Oracle GoldenGate

Learn how to prepare your database for Oracle GoldenGate, including how to configure connections and logging, how to enable Oracle GoldenGate in your database, how to set the flashback query, and how to manage server resources.

Topics:

- Enabling Oracle GoldenGate in the Database
  The database services required to support Oracle GoldenGate capture and apply must be enabled explicitly for all Oracle database versions. This is required for Extract and all Replicat modes.
- Configuring Connections for Extract and Replicat Processes
  Extract and Replicat require a dedicated server connection in the `tnsnames.ora` file.
- Setting Flashback Query
  To process certain update records, Extract fetches additional row data from the source database.
- Managing Server Resources
  Extract interacts with an underlying logmining server in the source database and Replicat interacts with an inbound server in the target database. This section provides guidelines for managing the shared memory consumed by the these servers.
- Ensuring Row Uniqueness in Source and Target Tables
  Oracle GoldenGate requires a unique row identifier on the source and target tables to locate the correct target rows for replicated updates and deletes.
- Configuring Logging Properties
  Oracle GoldenGate relies on the redo logs to capture the data that it needs to replicate source transactions.
- Using the Extract Automated Capture Mode
  The automated capture (auto capture) mode allows automatically capturing the tables that have been enabled for Oracle GoldenGate auto capture.
- Additional Oracle GoldenGate Configuration for Your Database

Enabling Oracle GoldenGate in the Database

The database services required to support Oracle GoldenGate capture and apply must be enabled explicitly for all Oracle database versions. This is required for Extract and all Replicat modes.

To enable Oracle GoldenGate, set the following database initialization parameter. All instances in Oracle RAC must have the same setting.

```
ENABLE_GOLDENGATE_REPLICATION=true
```

This parameter alters the `DBA_FEATURE_USAGE_STATISTICS` view. For more information about this parameter, see Initialization Parameters.

Configuring Connections for Extract and Replicat Processes

Extract and Replicat require a dedicated server connection in the `tnsnames.ora` file.

Before you begin, make sure that there is a dedicated user on the Oracle database side, with the required privileges. See Granting User Privileges for Oracle Database 21c and Lower.

On the Oracle GoldenGate side, you direct the Extract and Replicat processes to use these connections by specifying the values for `USERID` or `USERIDALIAS` parameter in the Extract and Replicat parameter files. See Quickstart Your Data Replication with Oracle GoldenGate Microservices Architecture for exmaples of setting the credentials for connecting to the database instances from Oracle GoldenGate.

The following are the security options for specifying the connection string in the Extract or Replicat parameter file.

Credential store method:

```
USERIDALIAS ggeast
```

In the case of `USERIDALIAS`, the alias `ggeast` is stored in the Oracle GoldenGate credential store with the actual connection string. The following example uses the `INFO CREDENTIALSTORE` command to display the details of the credentials configured in Oracle GoldenGate:

```
INFO CREDENTIALSTORE DOMAIN OracleGoldenGate
```

Output:

```
Domain: OracleGoldenGate
  Alias: ggeast
  Userid: ggadmin@dc1.example.com:1521/DBEAST.example.com
```

Setting up a Bequeath connection

Oracle GoldenGate can connect to a database instance without using the network listener if a Bequeath connect descriptor is added in the `tnsnames.ora`.

The following example shows the configuration for connecting to a database using Bequeath connect descriptor:

```
dbbeq =  (DESCRIPTION=
      (ADDRESS=(PROTOCOL=beq)
          (ENVS='ORACLE_SID=sales,ORACLE_HOME=/app/db_home/
oracle,LD_LIBRARY_PATH=/app/db_home/oracle/lib')
          (PROGRAM=/app/db_home/oracle/bin/oracle)
      (ARGV0=oraclesales)
      (ARGS='(DESCRIPTION=(LOCAL=YES)(ADDRESS=(PROTOCOL=beq)))'))
        (CONNECT_DATA=(SID=sales)))
```

In this example:

`/app/db_home` is the target Oracle database installation directory

`sales` is the database service name

The `ORACLE_SID`, `ORACLE_HOME`, and `LD_LIBRARY_PATH` in the `ENVS` parameter refers to the target.

Note:

Make sure that there is no white space between these environment variable settings.

- Granting User Privileges for Oracle Database 21c and Lower

Granting User Privileges for Oracle Database 21c and Lower

The user privileges that are required for connecting to Oracle database from Oracle GoldenGate depend on the type of user.

The user privileges that are required for connecting to Oracle database from Oracle GoldenGate depend on the type of user.

Privileges should be granted depending on the actions that the user needs to perform as the GoldenGate Administrator User on the source and target databases. For example, to grant DML operation privileges to insert, update, and delete transactions to a user, use the `GRANT ANY INSERT`/`UPDATE`/`DELETE` privileges and to further allow users to work with tables and indexes as part of DML operations, use the `GRANT CREATE`/`DROP`/`ALTER ANY TABLE`/`INDEX` privileges.

If the GoldenGate Administrator user has the DBA role, additional object privileges are not needed. However, there might be security constraints granting the DBA role to the GoldenGate Administration user. The DBA role is not necessarily required for Oracle GoldenGate.

If there are many objects being replicated, you might consider using the ANY privilege for DML and DDL operations. This simplifies the provision of privileges to the GoldenGate Administrator users, as you only need to grant a few privileges depending on the database operations.

The following table describes some of the essential privileges for GoldenGate Administrator user for Oracle database. For explanation purposes, the table uses `c##ggadmin` as an example of a common user for a multitenant container database and `ggadmin` as the pluggable database (PDB) user. `PDBEAST` and `PDBWEST` are used as examples of PDB names.

The following table describes the essential privileges for GoldenGate Administrator user for using Oracle GoldenGate with on source and target Oracle databases:

Privilege	Extract	Replicat All Modes	Purpose
`RESOURCE`	Yes	Yes	Required to create objects In Oracle Database 12cR1 and later, instead of `RESOURCE`, grant the following privilege: `ALTER USER user QUOTA {size | UNLIMITED} ON tablespace;`
`CONNECT`	Yes	Yes	Common user `SYSTEM` connects to the root container. This privilege is essential when the DBA role is not assigned to the user. See an example of Example: Grant privileges using the DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE package.
`CREATE SESSION`	Yes	Yes	Required to connect to the database.
`CREATE VIEW`	Yes	Yes	Required to add the heartbeat table view. If you want to be specific to each object, you can also provide the privileges for each object individually. You may consider creating a specific database role to maintain such privileges.
`ALTER SYSTEM`	Yes	Yes	Perform administrative changes, such as enabling logging.
`ALTER USER`	Yes	Yes	Required for multitenant architecture and *`GGADMIN`* should be a valid Oracle GoldenGate administrator schema.

Privilege	Extract	Replicat All Modes	Purpose
`EXEC DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE ('REPUSER', CONTAINER=>'PDBEAST');`	Yes	Yes	• Required for Oracle Autonomous Database Extract and Replicat. Extracts in the root container (`CDB$ROOT`)) might require a value of `ALL` or a specific PDB (example: `pdbeast`). • Grant privileges for Extract and Replicat users. See Example: Grant privileges using the DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE package. • Grant privilges to capture from Virtual Private Database • Grants privilges to capture redacted data
Grant `DV_GOLDENGATE_ADMIN` and `DV_GOLDENGATE_REDO_ACCESS` privileges connected as `SYS` user to the Extract and the Replicat user.	Yes	Yes	Capture from Data Vault. See Privileges for Capturing from Oracle Data Vault.
Grant Replicat privileges in `DBMS_MACADM.ADD_AUTH_TO_REALM` if applying to a realm.	NA	Yes	Capture from Data Vault. See Privileges for Capturing from Oracle Data Vault.
`INSERT`, `UPDATE`, `DELETE` on target tables	NA	Yes	Apply replicated DML to target objects. See Details of Support for Objects and Operations in Oracle DML

`GRANT INSERT ANY TO...`
 	
`GRANT UPDATE ANY TO...`
 	
`GRANT DELETE ANY TO...`
	NA	Yes	Grant these privileges to the Replicat user, instead of granting `INSERT`, `UPDATE`, `DELETE` to every table, if replicating every table.

If DDL replication is performed, grant the following as Database Vault owner: `EXECUTE DBMS_MACADM.AUTHORIZE_DDL('`*`GGADMIN USER`*`', '`*`SCHEMA FOR DDL`*`');`	No	No	Capture from Data Vault. See Privileges for Capturing from Oracle Data Vault.

Privilege	Extract	Replicat All Modes	Purpose
DDL privileges on target objects (if using DDL support)	NA	Yes	Issue replicated DDL on target objects. See Details of Support for Objects and Operations in Oracle DDL.
`GRANT [CREATE|ALTER|DROP] ANY [TABLE|INDEX|VIEW|PROCEDURE] to GGADMIN;`	Yes	Yes	Grants privileges for DDL Replication for tables.
`CREATE ANY TABLE`	Yes	Yes	Grants privileges for creating table in any schema. To allow creating tables only in a specific schema, use the `CREATE TABLE` privilege.
`CREATE ANY VIEW`	Yes	Yes	Grants privilges to create view in any database schema. To allow creating views in a specific schema, use the `CREATE VIEW` privilege.
`SELECT ANY DICTIONARY`	Yes	Yes	Allow all privileges to work properly on dictionary tables.

Example: Permissions granted for the Oracle database common user

Privilges granted for the Oracle database common user, which is `c##ggadmin` in the following example:

```
CREATE USER c##ggadmin IDENTIFIED BY passw0rd CONTAINER=all DEFAULT
TABLESPACE GG_DATA TEMPORARY TABLESPACE temp;
GRANT RESOURCE to c##ggadmin;
GRANT CREATE SESSION to c##ggadmin;
GRANT CREATE VIEW to c##ggadmin;
GRANT CREATE TABLE to c##ggadmin;
GRANT CONNECT to c##ggadmin CONTAINER=all;
GRANT DV_GOLDENGATE_ADMIN; --- for data vault user
GRANT DV_GOLDENGATE_REDO_ACCESS; --- for data vault user
GRANT ALTER SYSTEM to c##ggadmin;
GRANT ALTER USER to c##ggadmin;
ALTER USER c##ggadmin SET CONTAINER_DATA=all CONTAINER=current;
ALTER USER c##ggadmin QUOTA unlimited ON GG_DATA;
GRANT SELECT ANY DICTIONARY to c##ggadmin;
GRANT SELECT ANY TRANSACTION to c##ggadmin;
EXEC DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE('c##ggadmin');
```

In this example, DBA privilege is not provided but the user will be able to access the `DBA_SYS_PRIVS` package, if required.

Privileges granted for PDB user `ggadmin` are provided in the following example:

Example: Grant privileges using the DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE package

This procedure grants the privileges needed by a user to be an Oracle GoldenGate administrator The following example grants explicit privileges for Extract on Oracle multitenant database:

```
BEGIN
DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE
(GRANTEE => 'c##ggadmin',  PRIVILEGE_TYPE => 'CAPTURE',
 GRANT_SELECT_PRIVILEGES => TRUE,  DO_GRANTS => TRUE,  CONTAINER => 'ALL'
 );
END;
```

See `DBMS_GOLDENGATE_AUTH` in *Oracle Database PL/SQL Packages and Types Reference* for more information.

- Privileges for Capturing from Oracle Data Vault

Privileges for Capturing from Oracle Data Vault

Grant the following privileges connected as `SYS` user in Oracle database. These privileges are set for Extract and Replicat user credentials:

-
  ```
  EXEC DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE ('userID','*',
  GRANT_OPTIONAL_PRIVILEGES=>'*');
  GRANT DV_GOLDENGATE_ADMIN, DV_GOLDENGATE_REDO_ACCESS to userID;
  ```
- Grant Replicat the privileges in `DBMS_MACADM.ADD_AUTH_TO_REALM` if applying to a realm.

  Connect as Database Vault owner and execute the following scripts:

  ```
  BEGIN
  DVSYS.DBMS_MACADM.ADD_AUTH_TO_REALM(
  REALM_NAME => 'Oracle Default Component Protection Realm',GRANTEE =>
  'userID',AUTH_OPTIONS => 1) ;
  END ;
  /
  EXECUTE_DBMS_MACADM.AUTHORIZE_DDL('SYS', 'SYSTEM');
  ```
- For DDL replication, grant the following as the Database Vault owner:

  ```
  EXECUTE DBMS_MACADM.AUTHORIZE_DDL
  (‘userID', ‘SCHEMA FOR DDL’);
  ```

Setting Flashback Query

To process certain update records, Extract fetches additional row data from the source database.

Oracle GoldenGate fetches data for the following:

- User-defined types
- Nested tables
- XMLType objects

By default, Oracle GoldenGate uses Flashback Query to fetch the values from the undo (rollback) tablespaces. That way, Oracle GoldenGate can reconstruct a read-consistent row image as of a specific time or SCN to match the redo record.

For best fetch results, configure the source database as follows:

1. Set a sufficient amount of redo retention by setting the Oracle initialization parameters `UNDO_MANAGEMENT` and `UNDO_RETENTION` as follows (in seconds).

   ```
   UNDO_MANAGEMENT=AUTO

   UNDO_RETENTION=86400

   UNDO_RETENTION can be adjusted upward in high-volume environments.
   ```

2. Calculate the space that is required in the undo tablespace by using the following formula.

   ```
   undo_space = UNDO_RETENTION * UPS + overhead
   ```

   Where:

   - *`undo_space`* is the number of undo blocks.
   - `UNDO_RETENTION` is the value of the `UNDO_RETENTION` parameter (in seconds).
   - *`UPS`* is the number of undo blocks for each second.
   - *`overhead`* is the minimal overhead for metadata (transaction tables, etc.).

   Use the system view `V$UNDOSTAT` to estimate *`UPS`* and *`overhead`*.

3. For tables that contain `LOB`s, do one of the following:

   - Set the `LOB` storage clause to `RETENTION`. This is the default for tables that are created when `UNDO_MANAGEMENT` is set to `AUTO`.
   - If using `PCTVERSION` instead of `RETENTION`, set `PCTVERSION` to an initial value of 25. You can adjust it based on the fetch statistics that are reported with the `STATS EXTRACT` command. If the value of the `STAT_OPER_ROWFETCH CURRENTBYROWID` or `STAT_OPER_ROWFETCH_CURRENTBYKEY` field in these statistics is high, increase `PCTVERSION` in increments of 10 until the statistics show low values.

4. Grant either of the following privileges to the Oracle GoldenGate Extract user:

   ```
   GRANT FLASHBACK ANY TABLE TO db_user

   GRANT FLASHBACK ON schema.table TO db_user
   ```

Oracle GoldenGate provides the following parameters to manage fetching.

Parameter or Command	Description
`STATS EXTRACT` command with `REPORTFETCH` option	Shows Extract fetch statistics on demand.

Parameter or Command	Description
`STATOPTIONS` parameter with `REPORTFETCH` option	Sets the `STATS EXTRACT` command so that it always shows fetch statistics.
`MAXFETCHSTATEMENTS` parameter	Controls the number of open cursors for prepared queries that Extract maintains in the source database, and also for `SQLEXEC` operations.
`MAXFETCHSTATEMENTS` parameter	Controls the default fetch behavior of Extract: whether Extract performs a flashback query or fetches the current image from the table.
`FETCHOPTIONS` parameter with the `USELATESTVERSION` or `NOUSELATESTVERSION` option	Handles the failure of an Extract flashback query, such as if the undo retention expired or the structure of a table changed. Extract can fetch the current image from the table or ignore the failure.
`REPFETCHEDCOLOPTIONS` parameter	Controls the response by Replicat when it processes trail records that include fetched data or column-missing conditions.

Managing Server Resources

Extract interacts with an underlying logmining server in the source database and Replicat interacts with an inbound server in the target database. This section provides guidelines for managing the shared memory consumed by the these servers.

The shared memory that is used by the servers comes from the Streams pool portion of the System Global Area (SGA) in the database. Therefore, you must set the database initialization parameter `STREAMS_POOL_SIZE` high enough to keep enough memory available for the number of Extract and Replicat processes that you expect to run in integrated mode. Note that Streams pool is also used by other components of the database (like Oracle Streams, Advanced Queuing, and Datapump export/import), so make certain to take them into account while sizing the Streams pool for Oracle GoldenGate.

By default, one Extract requests the logmining server to run with `MAX_SGA_SIZE` of 1GB. Thus, if you are running three Extracts in the same database instance, you need at least 3 GB of memory allocated to the Streams pool. As a best practice, keep 25 percent of the Streams pool available. For example, if there are 3 Extracts, set `STREAMS_POOL_SIZE` for the database to the following value:

```
3 GB * 1.25 = 3.75 GB
```

Ensuring Row Uniqueness in Source and Target Tables

Oracle GoldenGate requires a unique row identifier on the source and target tables to locate the correct target rows for replicated updates and deletes.

Unless a `KEYCOLS` clause is used in the `TABLE` or `MAP` statement, Oracle GoldenGate selects a row identifier to use in the following order of priority, depending on the number and type of constraints that were logged (see Configuring Logging Properties).

1. Primary key if it does not contain any extended (32K) `VARCHAR2/NVARCHAR2` columns. Primary key without invisible columns.
2. Unique key: Unique key without invisible columns.

In the case of a non-integrated Replicat, the selection of the unique key is as follows:

- First unique key alphanumerically with no virtual columns, no UDTs, no function-based columns, no nullable columns, and no extended (32K) `VARCHAR2/NVARCHAR2` columns. To support a key that contains columns that are part of an invisible index, you must use the `ALLOWINVISIBLEINDEXKEYS` parameter in the Oracle GoldenGate `GLOBALS` file.
- First unique key alphanumerically with no virtual columns, no UDTs, no extended (32K) `VARCHAR2/NVARCHAR2` columns, or no function-based columns, but can include nullable columns. To support a key that contains columns that are part of an invisible index, you must use the `ALLOWINVISIBLEINDEXKEYS` parameter in the Oracle GoldenGate `GLOBALS` file.

3. Not Nullable Unique keys: At least one column from one of the unique keys must be not nullable. This is because `NOALLOWNULLABLEKEYS` is the default.

   Note:

   `ALLOWNULLABLEKEYS` is not valid for integrated Replicat.

4. If none of the preceding key types exist (even though there might be other types of keys defined on the table) Oracle GoldenGate constructs a pseudo key of all columns that the database allows to be used in a unique key, excluding virtual columns, UDTs, function-based columns, extended (32K) `VARCHAR2/NVARCHAR2` columns, and any columns that are explicitly excluded from the Oracle GoldenGate configuration by an Oracle GoldenGate user.

   Unless otherwise excluded due to the preceding restrictions, invisible columns are allowed in the pseudo key.

Note:

If there are other, non-usable keys on a table or if there are no keys at all on the table, Oracle GoldenGate logs an appropriate message to the report file. Constructing a key from all of the columns impedes the performance of Oracle GoldenGate on the source system. On the target, this key causes Replicat to use a larger, less efficient `WHERE` clause.

If a table does not have an appropriate key, or if you prefer the existing key(s) not to be used, you can define a substitute key if the table has columns that always contain unique values. You define this substitute key by including a `KEYCOLS` clause within the Extract `TABLE` parameter and the Replicat `MAP` parameter. The specified key will override any existing primary or unique key that Oracle GoldenGate finds. For more information, see *Reference for Oracle GoldenGate*.

Configuring Logging Properties

Oracle GoldenGate relies on the redo logs to capture the data that it needs to replicate source transactions.

The Oracle redo logs on the source system must be configured properly before you start Oracle GoldenGate processing.

This section addresses the following logging levels that apply to Oracle GoldenGate. The logging level that you use depends on Oracle GoldenGate features that you are using.

Note:

Redo volume is increased as the result of this required logging. You can wait until you are ready to start Oracle GoldenGate processing to enable the logging.

This table shows the Oracle GoldenGate use cases for the different logging properties.

Logging option	Command	What it does	Use case
Forced logging mode	`ALTER DATABASE FORCE LOGGING;`	Forces the logging of all transactions and loads.	Strongly recommended for all Oracle GoldenGate use cases. `FORCE LOGGING` overrides any table-level `NOLOGGING` settings.
Minimum database-level supplemental logging	`ALTER DATABASE ADD SUPPLEMENTAL LOG DATA`	Enables minimal supplemental logging to add row-chaining information to the redo log.	Required for all Oracle GoldenGate use cases
Schema-level supplemental logging, default setting See Enabling Schema-level Supplemental Logging.	`ADD SCHEMATRANDATA`	Enables unconditional supplemental logging of the primary key and conditional supplemental logging of unique key(s) and foreign key(s) of all tables in a schema. All of these keys together are known as the *scheduling columns*.	Enables the logging for all current and future tables in the schema. If the primary key, unique key, and foreign key columns are not identical at both source and target, use `ALLCOLS`. Required when using DDL support.

Logging option	Command	What it does	Use case
Schema-level supplemental logging with unconditional logging for all supported columns. (See Enabling Schema-level Supplemental Logging for non-supported column types.)	`ADD SCHEMATRANDATA` with `ALLCOLS` option	Enables unconditional supplemental logging of all of the columns in a table, for all of the tables in a schema.	Used for bidirectional and active-active configurations where all column values are checked, not just the changed columns, when attempting to perform an update or delete. This takes more resources though allows for the highest level of real-time data validation and thus conflict detection. This method should also be used if they are going to be using the `HANDLECOLLISIONS` parameter for initial loads.
Schema-level supplemental logging, minimal setting	`ADD SCHEMATRANDATA` with `NOSCHEDULINGCOLS` option	Enables unconditional supplemental logging of the primary key and all valid unique indexes of all tables in a schema.	Use only for nonintegrated Replicat. This is the minimum required schema-level logging.
Table-level supplemental logging with built-in support for integrated Replicat See Enabling Table-level Supplemental Logging	`ADD TRANDATA`	Enables unconditional supplemental logging of the primary key and conditional supplemental logging of unique key(s) and foreign key(s) of a table. All of these keys together are known as the *scheduling columns*.	Required for all Oracle GoldenGate use cases unless schema-level supplemental logging is used. If the primary key, unique key, and foreign key columns are not identical at both source and target, use `ALLCOLS`.
Table-level supplemental logging with unconditional logging for all supported columns. (See Enabling Table-level Supplemental Logging for non-supported column types.)	`ADD TRANDATA` with `ALLCOLS` option	Enables unconditional supplemental logging of all of the columns of the table.	Used for bidirectional and active-active configurations where all column values are checked, not just the changed columns, when attempting to perform an update or delete. This takes more resources though allows for the highest level of real-time data validation and thus conflict detection. It can also be used when the source and target primary, unique, and foreign keys are not the same or are constantly changing between source and target.

Logging option	Command	What it does	Use case
Table-level supplemental logging, minimal setting	`ADD TRANDATA` with `NOSCHEDULINGCOLS` option	Enables unconditional supplemental logging of the primary key and all valid unique indexes of a table.	Use for nonintegrated Replicat and non-parallel Replicat. This is the minimum required table-level logging.

Note:

Oracle Databases must be in `ARCHIVELOG` mode so that Extract can process the log files.

Topics:

- Enabling Subset Database Replication Logging
- Enabling Schema-level Supplemental Logging
- Enabling Table-level Supplemental Logging

Enabling Subset Database Replication Logging

Oracle strongly recommends putting the Oracle source database into forced logging mode. Forced logging mode forces the logging of all transactions and loads, overriding any user or storage settings to the contrary. This ensures that no source data in the Extract configuration gets missed.

There is a fine-granular database supplemental logging mode called Subset Database Replication available in LogMiner, which is the basic recommended mode for all Oracle GoldenGate and XStream clients. It replaces the previously used Minimum Supplemental Logging mode.

To know more, see `ALTER DATABASE`in the *Oracle Database SQL Language Reference*.

The subset database replication logging is enabled at `CDB$ROOT` (and all user-PDBs inherit it) currently.

Note:

Database-level primary key (PK) and unique index (UI) logging is only discouraged if you are replicating a subset of tables. You can use it with Live Standby, or if Oracle GoldenGate is going to replicate all tables, like to reduce the downtime for a migration or upgrade.

Perform the following steps to verify and enable, if necessary, subset database replication logging and forced logging.

1. Log in to SQL*Plus as a user with `ALTER SYSTEM` privilege.

2. Issue the following command to determine whether the database is in supplemental logging mode and in forced logging mode. If the result is `YES` for both queries, the database meets the Oracle GoldenGate requirement.

   ```
   SELECT SUPPLEMENTAL_LOG_DATA_MIN, FORCE_LOGGING FROM V$DATABASE;
   ```

3. If the result is `NO` for either or both properties, continue with these steps to enable them as needed:

   ```
   ALTER PLUGGABLE DATABASE pdbname ADD SUPPLEMENTAL LOG DATA SUBSET
   DATABASE REPLICATION;;
   ALTER DATABASE FORCE LOGGING;
   ```

4. Issue the following command to verify that these properties are now enabled.

   ```
   SELECT SUPPLEMENTAL_LOG_DATA_MIN, FORCE_LOGGING FROM V$DATABASE;
   ```

   The output of the query must be `YES` for both properties.

5. Switch the log files.

   ```
   ALTER SYSTEM SWITCH LOGFILE;
   ```

To perform dictionary validation, run the following command:

```
SELECT con_id, MINIMAL, SUBSET_REP, PRIMARY_KEY, UNIQUE_INDEX,
FOREIGN_KEY, ALL_COLUMN FROM CDB_SUPPLEMENTAL_LOGGING;
```

The output of this query should be `YES`.

```
SUBSET_REP = YES
```

For the following query:

```
SELECT NAME, LOG_MODE, FORCE_LOGGING, SUPPLEMENTAL_LOG_DATA_MIN,
SUPPLEMENTAL_LOG_DATA_PK PK, SUPPLEMENTAL_LOG_DATA_UI UI,
SUPPLEMENTAL_LOG_DATA_FK FK,
SUPPLEMENTAL_LOG_DATA_ALL,
SUPPLEMENTAL_LOG_DATA_SR FROM V$DATABASE;
```

For the query for `SUPPLEMENTAL_LOG_DATA_SR` the output should be `YES` and for `SUPPLEMENTAL_LOG_DATA_MIN` the output should be `IMPLICIT`.

To switch from earlier minimum supplemental logging to the new subset supplemental logging:

1. Drop the earlier higher levels on `CDB$ROOT`.

   ```
   ALTER DATABASE DROP SUPPLEMENTAL LOG DATA (PRIMARY KEY) COLUMNS;
   ALTER DATABASE DROP SUPPLEMENTAL LOG DATA (UNIQUE) COLUMNS;
   ALTER DATABASE DROP SUPPLEMENTAL LOG DATA;
   ```

2. Add only `subset database replication` mode:

   ```
   ALTER PLUGGABLE DATABASE pdbname ADD SUPPLEMENTAL LOG DATA SUBSET
   DATABASE REPLICATION;
   ```

3. Ensure that all PDBs inherit this `subset database replication` mode.

Enabling Schema-level Supplemental Logging

Oracle GoldenGate supports schema-level supplemental logging. Schema-level logging is required for an Oracle source database when using the Oracle GoldenGate DDL replication feature. In all other use cases, it is optional, but then you must use table-level logging instead (see Enabling Table-level Supplemental Logging).

By default, schema-level logging automatically enables unconditional supplemental logging of the primary key and conditional supplemental logging of unique key(s) and foreign key(s) of all tables in a schema. Options enable you to alter the logging as needed.

Note:

Oracle strongly recommends using schema-level logging rather than table-level logging, because it ensures that any new tables added to a schema are captured if they satisfy wildcard specifications. This method is also recommended because any changes to key columns are automatically reflected in the supplemental log data too. For example, if a key changes, there is no need to issue `ADD TRANDATA`.

Perform the following steps on the source system to enable schema-level supplemental logging.

1. Start the command line on the source system.
2. Issue the `DBLOGIN` command with the alias of a user in the credential store who has privilege to enable schema-level supplemental logging.

   ```
   DBLOGIN USERIDALIAS alias
   ```

   See `USERIDALIAS` in *Reference for Oracle GoldenGate* for more information about `USERIDALIAS` and additional options.

3. When using `ADD SCHEMATRANDATA` or `ADD TRANDATA` on a multitenant database, you can either log directly into the PDB and perform the command. Alternately, if you are logging in at the root level (using a `C##` user), then you must include the PDB. Issue the `ADD SCHEMATRANDATA` command for each schema for which you want to capture data changes with Oracle GoldenGate.

   ```
   ADD SCHEMATRANDATA pdb.schema [ALLCOLS | NOSCHEDULINGCOLS]
   ```

   Where:

   - Without options, `ADD SCHEMATRANDATA` schema enables the unconditional supplemental logging on the source system of the primary key and the conditional supplemental logging of all unique key(s) and foreign key(s) of all current and future tables in the given schema. Unconditional logging forces the primary key values to the log whether or not the key was changed in the current operation. Conditional logging logs all of the column values of a foreign or unique key if at least one of them was changed in the current operation. The default is optional to support nonintegrated Replicat but is required to support integrated Replicat because primary key, unique keys, and foreign keys must all be available to the inbound server to compute dependencies. For more information about integrated Replicat, see Deciding Which Apply Method to Use.

- `ALLCOLS` can be used to enable the unconditional supplemental logging of all of the columns of a table and applies to all current and future tables in the given schema. Use to support integrated Replicat when the source and target tables have different scheduling columns. (*Scheduling columns* are the primary key, the unique key, and the foreign key.)
- `NOSCHEDULINGCOLS` logs only the values of the primary key and all valid unique indexes for existing tables in the schema and new tables added later. This is the minimal required level of schema-level logging and is valid only for Replicat in nonintegrated mode.

In the following example, the command enables default supplemental logging for the `hr` schema.

```
ADD SCHEMATRANDATA pdbeast.hr ALLCOLS
```

In the following example, the command enables the supplemental logging only for the primary key and valid unique indexes for the `HR` schema.

```
ADD SCHEMATRANDATA pdbeast.hr NOSCHEDULINGCOLS
```

Enabling Table-level Supplemental Logging

Enable table-level supplemental logging on the source system in the following cases:

- To enable the required level of logging when not using schema-level logging (see Enabling Schema-level Supplemental Logging). Either schema-level or table-level logging must be used. By default, table-level logging automatically enables unconditional supplemental logging of the primary key and conditional supplemental logging of unique key(s) and foreign key(s) of a table. Options enable you to alter the logging as needed.
- To prevent the logging of the primary key for any given table.
- To log non-key column values at the table level to support specific Oracle GoldenGate features, such as filtering and conflict detection and resolution logic.
- If the key columns change on a table that only has table-level supplemental logging, you must perform `ADD TRANDATA` on the table prior to allowing any DML activity on the table.

Perform the following steps on the source system to enable table-level supplemental logging or use the optional features of the command.

1. Run the command line on the source system.

2. Issue the `DBLOGIN` command using the alias of a user in the credential store who has privilege to enable table-level supplemental logging.

   ```
   DBLOGIN USERIDALIAS alias
   ```

   See `USERIDALIAS` in *Reference for Oracle GoldenGate*for more information about `DBLOGIN` and additional options.

3. Issue the `ADD TRANDATA` command.

   ```
   ADD TRANDATA [PDB.]schema.table [, COLS (columns)] [, NOKEY] [, ALLCOLS |
   NOSCHEDULINGCOLS]
   ```

   Where:

   - *`PDB`* is the name of the root container or pluggable database if the table is in a multitenant container database.
   - *`schema`* is the source schema that contains the table.
   - *`table`* is the name of the table. See Specifying Object Names in Oracle GoldenGate Input in *Administering Oracle GoldenGate* for instructions for specifying object names.
   - `ADD TRANDATA` without other options automatically enables unconditional supplemental logging of the primary key and conditional supplemental logging of unique key(s) and foreign key(s) of the table. Unconditional logging forces the primary key values to the log whether or not the key was changed in the current operation. Conditional logging logs all of the column values of a foreign or unique key if at least one of them was changed in the current operation. The default is optional to support nonintegrated Replicat (see also `NOSCHEDULINGCOLS`) but is required to support integrated Replicat because primary key, unique keys, and foreign keys must all be available to the inbound server to compute dependencies. For more information about integrated Replicat, see Deciding Which Apply Method to Use.
   - `ALLCOLS` enables the unconditional supplemental logging of all of the columns of the table. Use to support integrated Replicat when the source and target tables have different scheduling columns. (*Scheduling columns* are the primary key, the unique key, and the foreign key.)
   - `NOSCHEDULINGCOLS` is valid for Replicat in nonintegrated mode only. It issues an `ALTER TABLE` command with an `ADD SUPPLEMENTAL LOG DATA ALWAYS` clause that is appropriate for the type of unique constraint that is defined for the table, or all columns in the absence of a unique constraint. This command satisfies the basic table-level logging requirements of Oracle GoldenGate when schema-level logging will not be used. See Ensuring Row Uniqueness in Source and Target Tables for how Oracle GoldenGate selects a key or index.
   - `COLS` *`columns`* logs non-key columns that are required for a `KEYCOLS` clause or for filtering and manipulation. The parentheses are required. These columns will be logged in addition to the primary key unless the `NOKEY` option is also present.
   - `NOKEY` prevents the logging of the primary key or unique key. Requires a `KEYCOLS` clause in the `TABLE` and `MAP` parameters and a `COLS` clause in the `ADD TRANDATA` command to log the alternate `KEYCOLS` columns.

4. If using `ADD TRANDATA` with the `COLS` option, create a unique index for those columns on the target to optimize row retrieval. If you are logging those columns as a substitute key

for a `KEYCOLS` clause, make a note to add the `KEYCOLS` clause to the `TABLE` and `MAP` statements when you configure the Oracle GoldenGate processes.

Using the Extract Automated Capture Mode

The automated capture (auto capture) mode allows automatically capturing the tables that have been enabled for Oracle GoldenGate auto capture.

See How to Capture Supplemental Logging for Oracle GoldenGate in the *Oracle Database Utilities* guide.

Here are some benefits of using the auto capture mode:

- Easy to configure captured table set
- No requirement to update `TABLE`/`TABLEEXCLUDE` parameter
- No need to stop or restart Extract when captured table set changes

Enabling Extract Auto Capture Mode

To enable the Extract Auto Capture Mode, enable the auto capture mode using `TRANLOGOPTIONS`:

```
TRANLOGOPTIONS ENABLE_AUTO_CAPTURE | DISABLE_AUTO_CAPTURE
```

When Extract is running in the auto capture mode, don't filter an LCR if the object is not part of exclusion list set by `TABLE EXCLUDE` parameter or any inclusion list set by `TABLE` parameter.

The `LIST TABLES` command shows the list of tables enabled for `AUTO_CAPTURE`.

See DML Auto Capture and Supported Objects and Operations in Oracle DDL to know about the DML and DDL considerations.

Also see this article Oracle GoldenGate 21c: Auto Capture of Tables to learn more.

Additional Oracle GoldenGate Configuration for Your Database

This chapter contains additional configuration considerations that may apply to your database environment.
Topics:

- Installing Support for Oracle Sequences
  To support Oracle sequences, you must install some database procedures.
- Handling Other Database Properties
  This topic describes the database properties that may affect Oracle GoldenGate and the parameters that you can use to resolve or work around the condition.

Installing Support for Oracle Sequences

To support Oracle sequences, you must install some database procedures.

To Install Oracle Sequence Objects

1. In SQL*Plus, connect to the source and target Oracle systems as `SYSDBA`.
2. If you already assigned a database user to support the Oracle GoldenGate DDL replication feature, you can skip this step. Otherwise, in SQL*Plus on both systems create a database user that can also be the DDL user.

   ```
   CREATE USER DDLuser IDENTIFIED BY password;
   GRANT CONNECT, RESOURCE, DBA TO DDLuser;
   ```

3. From the Oracle GoldenGate installation directory on each system, run GGSCI.
4. In GGSCI, issue the following command on each system.

   ```
   EDIT PARAMS ./GLOBALS
   ```

5. In each `GLOBALS` file, enter the `GGSCHEMA` parameter and specify the schema of the DDL user that you created earlier in this procedure.

   ```
   GGSCHEMA schema
   ```

6. Save and close the files.
7. In SQL*Plus on the source system, issue the following statement in SQL*Plus.

   ```
   ALTER TABLE sys.seq$ ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY) COLUMNS;
   ```

To capture the sequence from a multitenant database

1. Create an Oracle GoldenGate user in each PDB that you need to capture sequences from.
2. Add the user to the GLOBALS parameter file. It is easier if you use the same user for each PDB, if you don't then you need to change the GLOBALS file each time you do step 3.
3. Log into Admin Client or GGSCI.
4. Connect to the root container on the source using `DBLOGIN`.
5. Issue the `FLUSH SEQUENCE` command for each PDB.

If you don't want to keep these database accounts, you can drop the user or deactivate the account.

Here is an example of the entire process:

```
Environment information
            OGG 19.1 Oracle 12c to Oracle 12c Replication, Integrated
            Extract, Parallel Replicat
            Source: CDB GOLD, PDB CERTMISSN
            Target: CDB PLAT, PDB CERTDSQ
```

```
            Source Oracle GoldenGate Configuration
            Container User: C##GGADMIN
            PDB User for Sequences: GGATE
```

When prompted, enter `GGATE`

```
GLOBALS
            GGSCHEMA GGATE
         Flush Sequence
            GGSCI> DBLOGIN USERIDALIAS GGADMIN DOMAIN GOLD_QC_CDB$ROOT
            GGSCI> FLUSH SEQUENCE CERTMISSN.SRCSCHEMA1.
Target OGG Configuration
         PDB User: GGATE
         Run @sequence
                sqlplus / as sysdba
                SQL> alter session set container=CERTDSQ;
                SQL> @sequence
```

When prompted enter `GGATE`.

Replicating sequences is required for High Availability (HA) and DR scenarios:

- For migrations, you need rebuild the sequences on the target during the switchover, or increase them to a higher value just prior to the switchover.
- Make sure you place the sequences into their own Replicat.

Handling Other Database Properties

This topic describes the database properties that may affect Oracle GoldenGate and the parameters that you can use to resolve or work around the condition.

The following table lists the database properties and the associated concern/ resolution.

Database Property	**Concern/Resolution**
Table with interval partitioning	To support tables with interval partitioning, make certain that the `WILDCARDRESOLVE` parameter remains at its default of `DYNAMIC`.
Table with virtual columns	Virtual columns are not logged, and Oracle does not permit DML on virtual columns. You can, however, capture this data and map it to a target column that is not a virtual column by doing the following: Include the table in the Extract `TABLE` statement and use the `FETCHCOLS` option of `TABLE` to fetch the value from the virtual column in the database. In the Replicat `MAP` statement, map the source virtual column to the non-virtual target column.
Table with inherently updateable view	To replicate to an inherently updateable view, define a key on the unique columns in the updateable view by using a `KEYCOLS` clause in the same `MAP` statement in which the associated source and target tables are mapped.
Redo logs or archives in different locations	The `TRANLOGOPTIONS` parameter contains options to handle environments where the redo logs or archives are stored in a different location than the database default or on a different platform from that on which Extract is running. For more information, see *Reference for Oracle GoldenGate*.

Database Property	Concern/Resolution
`TRUNCATE` operations	To replicate `TRUNCATE` operations, choose one of two options: • Standalone `TRUNCATE` support by means of the `GETTRUNCATES` parameter replicates `TRUNCATE TABLE`, but no other `TRUNCATE` options. Use only if not using Oracle GoldenGate DDL support. • The full DDL support replicates `TRUNCATE TABLE`, `ALTER TABLE TRUNCATE PARTITION`, and other DDL.
Sequences	To replicate DDL for sequences (`CREATE`, `ALTER`, `DROP`, `RENAME`), use Oracle GoldenGate DDL support. To replicate just sequence values, use the `SEQUENCE` parameter in the Extract parameter file. This does *not* require the Oracle GoldenGate DDL support environment. For more information, see *Reference for Oracle GoldenGate*.

2

Establishing Oracle GoldenGate Credentials

Learn how to create database users for the processes that interacts with the database, assign the correct privileges, and secure the credentials from unauthorized use.

Topics

Assigning Credentials to Oracle GoldenGate

The Oracle GoldenGate processes require one or more database credentials with the correct database privileges for the database version, database configuration, and Oracle GoldenGate features that you are using.

Create users for the source and target database instances, each one dedicated to Oracle GoldenGate. The assigned user can be the same user for all the Oracle GoldenGate processes that must connect to a source or target Oracle Database.

See `ALTER CREDENTIALSTORE` command usage to manage credentials in a credential store for Oracle GoldenGate users.

3

Configuring Oracle GoldenGate in a Multitenant Container Database

This chapter contains additional configuration instructions when configuring Oracle GoldenGate using the per-PDB capture mode or the CDB root capture mode.
Topics:

Requirements for Configuring Container Databases for Oracle GoldenGate

Here are configuration requirements to enable replication to and from multitenant container databases:

- The different pluggable databases in the multitenant container database can have different character sets. Oracle GoldenGate captures data from any multitenant database with different character sets into one trail file and replicates the data without corruption due to using different character sets.
- Oracle GoldenGate on-premise 21c and higher with Oracle Database 21c support the per-PDB-Extract feature. You can create a per-PDB Extract connecting to the local `ggadmin` user in a specific pluggable database to create and register the Extract. You do not need the container clause or the `SOURCECATALOG` to set up the per-PDB Extract. Setting up the Extract as a per-PDB-Extract matches exactly the same procedure as a Non-CDB.

  If required, you can also set up the root-level Extract connecting to the common `c##ggadmin` user in the root container to create and register the Extract for specific pluggable databases using the container. See Granting User Privileges for Oracle Database 21c and Lower depending on the Oracle database installation that you need to configure.

  See Establishing Oracle GoldenGate Credentials for how to create a user for the Oracle GoldenGate processes and grant the correct privileges.
- To support source CDB 12.2, Extract must specify the trail format as release 12.3. Due to changes in the redo logs, to capture from a multitenant database that is Oracle 12.2 or higher, the trail format release must be 12.3 or higher.

- The `dbms_goldengate_auth.grant_admin_privilege` package grants the appropriate privileges for capture and apply within a multitenant container database. This includes the `container` parameter, which must be set to `ALL`, as shown in the following example:

  ```
  EXCE DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE
  ('c##ggadmin',container=>'all')
  ```

- DDL replication works as a normal replication for multitenant databases. However, DDL on the root container should not be replicated because Replicats must not connect to the root container, only to PDBs.

You can also see Quickstart Your Data Replication with Oracle GoldenGate Microservices Architecture for a list of required privileges and how to configure the container and pluggable databases for Oracle GoldenGate.

- Flush Sequence for Multitenant Container Database

Flush Sequence for Multitenant Container Database

Use `FLUSH SEQUENCE` immediately after you start Extract for the first time during an initial synchronization or a re-synchronization. This command updates an Oracle sequence, so that initial redo records are available at the time that Extract starts to capture transaction data. Normally, redo is not generated until the current cache is exhausted. The flush gives Replicat an initial start point with which to synchronize to the correct sequence value on the target system. From then on, Extract can use the redo that is associated with the usual cache reservation of sequence values.

1. The following Oracle procedures are used by `FLUSH SEQUENCE`:

Database	Procedure	User and Privileges
Source	`updateSequence`	Grants `EXECUTE` to the owner of the Oracle GoldenGate DDL objects, or other selected user if not using DDL support.
Target	`replicateSequence`	Grants `EXECUTE` to the Oracle GoldenGate Replicat user.

   The `sequence.sql` script installs these procedures. Normally, this script is run as part of the Oracle GoldenGate installation process, but make certain that was done before using `FLUSH SEQUENCE`. If `sequence.sql` was not run, the flush fails and an error message similar to the following is generated:

   ```
   Cannot flush sequence {0}. Refer to the Oracle GoldenGate for
   Oracle
   documentation for instructions on how to set up and run the
   sequence.sql
   script. Error {1}.
   ```

2. The `GLOBALS` file must contain a `GGSCHEMA` parameter that specifies the schema in which the procedures are installed. This user must have `CONNECT`, `RESOURCE`, and `DBA` privileges.

3. Before using `FLUSH SEQUENCE`, issue the `DBLOGIN` command as the database user that has `EXECUTE` privilege on the `updateSequence` procedure. If logging into a multitenant container database, log into the pluggable database that contains the sequence that is to be flushed.

`FLUSH SEQUENCE` must be issued at the PDB level, to create an Oracle GoldenGate user in each PDB for which the sequence replication is required. Use `DBLOGIN` to log into that PDB, and run the `FLUSH SEQUENCE` command.

It is recommended that you use the same schema in each PDB, so that it works with the `GGSCHEMA` GLOBALS parameter file. Here is an example:

```
Environment Information OGG 21.3 Oracle 21c to Oracle 21c Replication,
Integrated Extract, Parallel Replicat
Source: CDB NORTH, PDB DBEAST
Target: CDB SOUTH, PDB DBWEST
Source OGG Configuration
    Container User: C##GGADMIN
    PDB User for Sequences: GGATE
sqlplus / as sysdbao
ALTER SESSION SESSION SET CONTAINER=CERTMISSN;
CREATE USER ggadmin CREATE USER ggate IDENTIFIED BY password DEFAULT
TABLESPACE USERS TEMPORARY TABLESPACE TEMP QUOTA UNLIMITED ON USERS
CONTAINER=CURRENT;


Run @sequence
sqlplus / as sysdba
SQL> ALTER SESSION SET CONTAINER=PDBEAST;
SQL> @sequence
```

When prompted enter

```
GGADMIN GLOBALS
GGSCHEMA GGADMIN


FLUSH SEQUENCE:

DBLOGIN USERIDALIAS ggeast DOMAIN OracleGoldenGate
FLUSH SEQUENCE PDBEAST.HR.*
```

Target Oracle GoldenGate Configuration:

```
 PDB User: ggadmin
Run @sequence
sqlplus / as sysdba
SQL> ALTER SESSION SET CONTAINER =PDBWEST;
SQL> @sequence
```

When prompted enter `ggadmin`.

This also applies to the `@sequence.sql script`, which must also be run on each PDB from where you are going to capture.

Using the Root Container Extract from PDB

Oracle GoldenGate 21c and higher releases allows capturing from a PDB in a multitenant container Oracle database. To capture from a multitenant database, you must use an Extract that is configured at the root level using a `c##` account. To apply data into a multitenant database, a separate Replicat is needed for each PDB, because a Replicat connects at the PDB level and doesn't have access to objects outside of that PDB

One Extract group can capture from multiple pluggable databases to a single trail. In the parameter file, source objects must be specified in `TABLE` and `SEQUENCE` statements with their fully qualified three-part names in the format of *`container.schema.object`*.

As an alternative to specifying three-part names, you can specify a default pluggable database with the `SOURCECATALOG` parameter, and then specify only the *`schema.object`* in subsequent `TABLE` or `SEQUENCE` parameters. You can use multiple instances of this configuration to handle multiple source pluggable databases. For example:

```
SOURCECATALOG DBEAST
TABLE hr.employees;
SEQUENCE hr.seq;
SOURCECATALOG DBWEST
TABLE hr.employees;
SEQUENCE hr.seq;
```

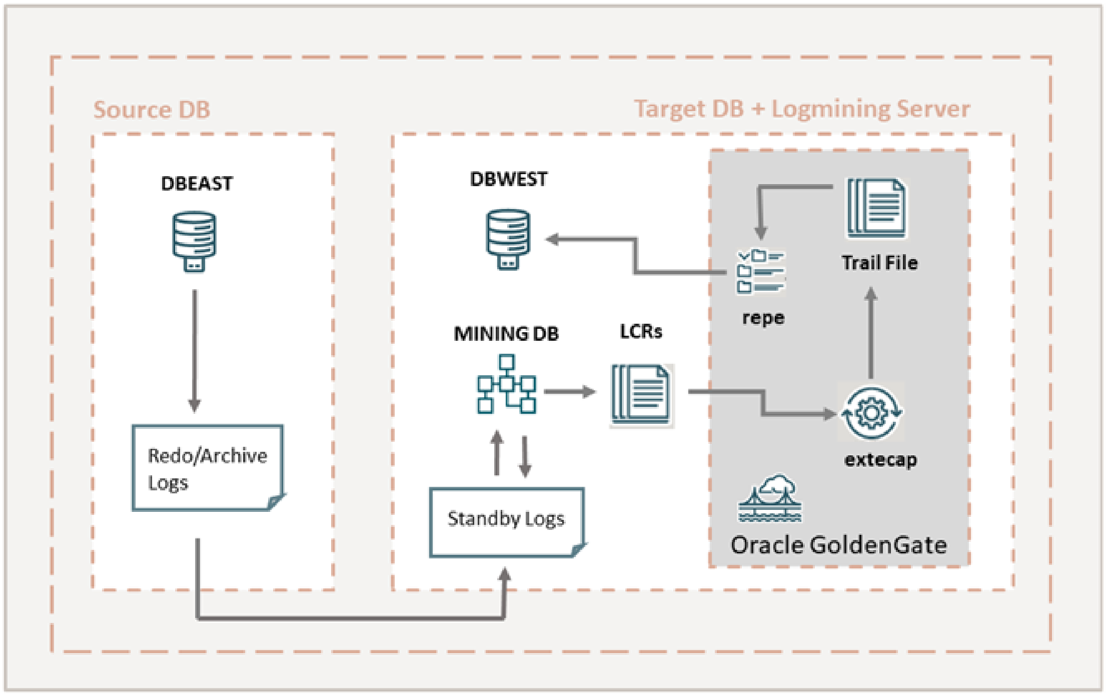
To learn about the steps for adding a per-PDB capture for Oracle database, see the following:

- Quickstart Bidirectional Replication
- Add Extracts
- About Extract
  The Oracle GoldenGate Extract process interacts directly with a database logmining server to receive data changes in the form of logical change records (LCRs).
- Configuring Primary Extract
  The mining database from which the primary Extract captures log change records from the logmining server, can be either local or downstream from the source database.
- Setting up the Auto-Capture Mode
- Mining Mode Toggling

About Extract

The Oracle GoldenGate Extract process interacts directly with a database logmining server to receive data changes in the form of logical change records (LCRs).

The following diagram illustrates the configuration of Extract.



Some of the additional features of Oracle GoldenGate Extract are:

- Extract is fully integrated with the database, allowing seamless interoperability between features such as Oracle RAC, ASM, and TDE.
- Extract uses the database logmining server to access the Oracle redo stream, with the benefit of being able to automatically switch between different copies of archive logs or different mirrored versions of the online logs. Thus, capture can transparently handle the absence of a log file caused by disk corruption, hardware failure, or operator error, assuming that additional copies of the archived and online logs are available
- Extract enables faster filtering of tables.
- Extract handles point-in-time recovery and RAC integration more efficiently.
- Extract features integrated log management. The Oracle Recovery Manager (RMAN) automatically retains the archive logs that are needed by Extract.
- Extract supports capture from a multitenant container database and from per-PDB capture mode.
- Extract and Replicat (integrated) are both database objects, so the naming of the objects follow the same rules as other Oracle database objects. See Specifying Object Names in Oracle GoldenGate Input in *Administering Oracle GoldenGate*.
- When Extract is running from a remote system, Oracle GoldenGate automatically enables cross endian interoperability. This implies that if the endian value where Extract is running is different from the endian value where the Oracle database is running, then the cross endian support is automatically enabled. For cross endian Extract to work, the compatibility parameter of the source database must be 11.2.0.4 or higher.
- Extract Deployment Options

Extract Deployment Options

The deployment options for Extract are described in this section and depend on where the mining database is deployed. The mining database is the one where the logmining server is deployed.

- **Local deployment**: For a local deployment, the source database and the mining database are the same. The source database is the database for which you want to mine the redo stream to capture changes, and also where you deploy the logmining server. Because integrated capture is fully integrated with the database, this mode does not require any special database setup.
- **Downstream deployment**: In a downstream deployment, the source and mining databases are different databases. You create the logmining server at the downstream database. You configure redo transport at the source database to ship the redo logs to the downstream mining database for capture at that location. Using a downstream mining server for capture may be desirable to offload the capture overhead and any other overhead from transformation or other processing from the production server, but requires log shipping and other configuration.

  When using a downstream mining configuration, the source database and mining database must be of the same platform. For example, if the source database is running on Windows 64-bit, the downstream database must also be on a Windows 64-bit platform. See Configuring a Downstream Mining Database to configure a downstream mining database.
- **Downstream sourceless Extract deployment:** In the Extract parameter file, replace the `USERID` parameter with `NOUSERID`. You must use `TRANLOGOPTIONS MININGUSER`. Extract obtains all required information from the downstream mining database. Extract is not dependent on any connection to the source database. The source database can be shutdown and restarted without affecting Extract.

  Extract will abend if it encounters redo changes that require data to be fetched from the source database.

  To capture any tables that are listed as `ID KEY` in the `dba_goldengate_support_mode` view, you need to have a `FETCHUSERID` or `FETCHUSERIDALIAS` connection to support the tables. Tables that are listed as `FULL` do not require this. We also need to state that if a customer wants to perform `SQLEXEC` operations that perform a query or execute a stored procedure they cannot use this method as it is incompatible with `NOUSERID` because `SQLEXEC` works with `USERID` or `USERIDALIAS`.

Configuring Primary Extract

The mining database from which the primary Extract captures log change records from the logmining server, can be either local or downstream from the source database.

These steps configure the primary Extract to capture transaction data from either location. See Configuring a Downstream Mining Database and see the following examples:

1. Example 1: Capturing from One Source Database in Real-time Mode
2. Example 2: Capturing from Multiple Sources in Archive-log-only Mode

3. Example 3: Capturing from Multiple Sources with Mixed Real-time and Archive-log-only Mode

> **Note:**
>
> One Extract group is generally sufficient to capture from a single database or multiple pluggable databases within a multitenant container database. See Configuring Oracle GoldenGate in a Multitenant Container Database . You can also choose per-PDB capture mode when working in an Oracle Autonomous Database or cloud environment. See Configuring Extract to Capture from an Autonomous Database.

1. In GGSCI, Admin Client, or REST API client on the source system, create the Extract parameter file.

   ```
   EDIT PARAMS name
   ```

   Where: *name* is the name of the primary Extract.

   > **Note:**
   >
   > To learn about using Oracle GoldenGate microservices to perform this task, see How to Add Extracts.

2. Enter the Extract parameters in the order shown, starting a new line for each parameter statement. Examples are shown for a regular database, a multitenant container database, and downstream deployments for both non-CDB and multitenant databases. The difference between the two is whether you must use two-part or three-part object names in the `TABLE` and `SEQUENCE` specifications. See the basic parameters for primary Extract for more information and parameter descriptions.

   Basic parameters for Extract mining a non-mulitenant database

   ```
   EXTRACT financep
   USERIDALIAS c##_alias
   DDL INCLUDE MAPPED
   EXTTRAIL /ggs/dirdat/lt
   SEQUENCE hr.employees_seq;
   TABLE hr.*;
   ```

   Basic parameters for Extract capturing from a multitenant database

   ```
   EXTRACT financep
   USERIDALIAS c##_alias
   DDL INCLUDE MAPPED
   EXTTRAIL /ggs/dirdat/lt
   TABLE test.ogg.tab1;
   SEQUENCE hr.employees_seq;
   TABLE hr.*;
   ```

```
TABLE sales.*;
TABLE acct.*;
```

Basic parameters for Extract where the mining database is a downstream database and is a non-CDB database

```
EXTRACT financep
USERIDALIAS c##_alias
TRANLOGOPTIONS MININGUSERALIAS c##_alias
TRANLOGOPTIONS INTEGRATEDPARAMS (DOWNSTREAM_REAL_TIME_MINE Y)
LOGALLSUPCOLS
UPDATERECORDFORMAT COMPACT
DDL INCLUDE MAPPED
ENCRYPTTRAIL AES192
EXTTRAIL /ggs/dirdat/lt
SEQUENCE hr.employees_seq;
TABLE hr.*;
```

Basic parameters for the primary Extract where the mining database is a downstream database and is a multitenant container database

```
EXTRACT financep
USERIDALIAS tiger1
TRANLOGOPTIONS MININGUSERALIAS tiger2
TRANLOGOPTIONS INTEGRATEDPARAMS (MAX_SGA_SIZE 164, &
   DOWNSTREAM_REAL_TIME_MINE y)
LOGALLSUPCOLS
UPDATERECORDFORMAT COMPACT
DDL INCLUDE MAPPED SOURCECATALOG pdb1 INCLUDE MAPPED SOURCECATALOG
pdb2
ENCRYPTTRAIL AES192EXTTRAIL /ggs/dirdat/lt
TABLE test.ogg.tab1;
SOURCECATALOG pdb1
SEQUENCE hr.employees_seq;
TABLE hr.*;
SOURCECATALOG pdb2
TABLE sales.*;
TABLE acct.*;
```

Parameter	Description
`EXTRACT` *`group`*	*`group`* is the name of the Extract group. For more information, see *Reference for Oracle GoldenGate*.
`USERIDALIAS` *`alias`*	Specifies the alias of the database login credential of the user that is assigned to Extract. This credential must exist in the Oracle GoldenGate credential store.

Parameter	Description
`LOGALLSUPCOLS`	Writes all supplementally logged columns to the trail, including those required for conflict detection and resolution and the scheduling columns required to support integrated Replicat. (Scheduling columns are primary key, unique index, and foreign key columns.) You configure the database to log these columns with GGSCI commands. See Establishing Oracle GoldenGate Credentials.
`UPDATERECORDFORMAT COMPACT`	Combines the before and after images of an `UPDATE` operation into a single record in the trail. This parameter is valid for Oracle Databases version 12c and later to support Replicat in integrated mode. Although not a required parameter, `UPDATERECORDFORMAT COMPACT` is a best practice and significantly improves Replicat performance.
`TRANLOGOPTIONS MININGUSERALIAS` *`alias`*	Specifies connection information for the logmining server at the downstream mining database, if being used. `MININGUSERALIAS` specifies the alias of the Extract user for the downstream mining database. This is the user that you created in Configuring a Downstream Mining Database . The credential for this user must be stored in the Oracle GoldenGate credential store. Use `MININGUSERALIAS` only if the database logmining server is in a different database from the source database; otherwise just use `USERIDALIAS`. When using `MININGUSERALIAS`, use it in addition to `USERIDALIAS`, because credentials are required for both databases.
`TRANLOGOPTIONS [INTEGRATEDPARAMS (`*`parameter`*`[, ...])]`	Optional, passes parameters to the Oracle Database that contains the database logmining server. Use only to change logmining server parameters from their default settings. See Additional Parameter Options for Extract.
`TRANLOGOPTIONS CHECKPOINTRETENTIONTIME` *`days`*	Optional, controls the number of days that Extract retains checkpoints before purging them automatically. Partial days can be specified using decimal values. For example, 8.25 specifies 8 days and 6 hours. For more information, see *Reference for Oracle GoldenGate*.
`DDL` *`include_clause`*	Required if replicating DDL operations. See Configuring DDL Support for more information.
`ENCRYPTTRAIL` *`algorithm`*	Encrypts the local trail.
`EXTTRAIL` *`pathname`*	Specifies the path name of the local trail to which the primary Extract writes captured data.
`SOURCECATALOG` *`container`*	Use this parameter when the source database is a multitenant container database. Specifies the name of a pluggable database that is to be used as the default container for all subsequent `TABLE` and `SEQUENCE` parameters that contain two-part names. This parameter enables you to use two-part object names (*`schema.object`*) rather than three-part names (*`container.schema.object`*). It remains in effect until another `SOURCECATALOG` parameter is encountered or a full three-part `TABLE` or `SEQUENCE` specification is encountered.

Parameter	Description
`{TABLE | SEQUENCE}` `[container.]schema.object;`	Specifies the database object for which to capture data. • `TABLE` specifies a table or a wildcarded set of tables. • `SEQUENCE` specifies a sequence or a wildcarded set of sequences. • *`container`* is the name of the pluggable database (PDB) that contains the object, if this database is a multitenant container database. The container part of the name is not required if this Extract group will only process data from one PDB and the default PDB is specified with the `SOURCECATALOG` parameter. • *`schema`* is the schema name or a wildcarded set of schemas. • *`object`* is the table or sequence name, or a wildcarded set of those objects. Terminate the parameter statement with a semi-colon. To exclude a name from a wildcard specification, use the `CATALOGEXCLUDE`, `SCHEMAEXCLUDE`, `TABLEEXCLUDE`, and `EXCLUDEWILDCARDOBJECTSONLY` parameters as appropriate.

`MAPINVISIBLECOLUMNS`	Controls whether or not Replicat includes invisible columns in Oracle target tables for default column mapping. Configure the invisible columns in your column mapping using SQL to explicitly specify column names. For example:
	`CREATE TABLE tab1 (id NUMBER, data CLOB INVISIBLE);`
	`   INSERT INTO tab1 VALUES (1, 'a');ERROR: ORA-913`
	`   INSERT INTO tab1 (id, data) VALUES (1, 'a'); OK`
	You can change the column visibility using `ALTER TABLE`. The invisible column can be part of an index, including primary key and unique index.

3. Enter any optional Extract parameters that are recommended for your configuration. You can edit this file at any point before starting processing by using the `EDIT PARAMS` command in GGSCI.
4. Save and close the file.

- Add the Local Trail
  These steps add the local trail to which the primary Extract writes captured data.
- Add the Remote Trail
  Although it is read by Replicat, this trail must be associated with the Extract, so it must be added on the source system, not the target.

Add the Local Trail

These steps add the local trail to which the primary Extract writes captured data.

On the source system, issue the `ADD EXTTRAIL` command on the command line:

```
ADD EXTTRAIL pathname, EXTRACT group name
```

Where:

- `EXTTRAIL` specifies that the trail is to be created on the local system.
- *`pathname`* is the relative or fully qualified name of the trail, including the two-character name.
- `EXTRACT` *`group name`* is the name of the primary Extract group.

> **Note:**
>
> Oracle GoldenGate creates this trail automatically during processing.

Example 3-1

```
ADD EXTTRAIL /north/ea, EXTRACT exte
```

Add the Remote Trail

Although it is read by Replicat, this trail must be associated with the Extract, so it must be added on the source system, not the target.

These steps add the remote trail:

On the source system, issue the following command:

```
ADD RMTTRAIL pathname, EXTRACT group name
```

Where:

- `RMTTRAIL` specifies that the trail is to be created on the target system.
- pathname is the relative or fully qualified name of the trail, including the two-character name.
- `EXTRACT` *`group name`* is the name of the data-pump Extract group.

> **Note:**
>
> Oracle GoldenGate creates this trail automatically during processing.

Example 3-2

```
ADD RMTTRAIL /south/re, EXTRACT exts
```

Setting up the Auto-Capture Mode

The automatic Extract mode captures changes for all the tables that are enabled for logical replication.

The auto-capture mode is available from Oracle Database 21c and higher. A table is enabled for logical replication or auto-capture when:

- It has sufficient ID or scheduling-key supplemental log data at table or schema level.
- It has primary key (PK), unique identifier (UI), foreign key (FK) supplemental log data, and ALLKEYS supplemental log data. ALLKEYS is required in addition to PK, UI and FK because it logs all unique keys at the schema-wide supplemental logging level in the absence of a primary key.

Benefits of Using the Auto Extract Mode

- Easy to configure captured table set
- When captured table set changes, you don't need to update the `TABLE`/ `TABLEEXCLUDE` parameter, or stop and restart Extract.

Enabling Auto Capture

See `TRANLOGOPTIONS` for syntax and usage.

Use the following DDLs to enable auto capture at the table level:

```
CREATE/ALTER TABLE table_name ENABLE LOGICAL REPLICATION ALLKEYS
```

or

```
CREATE/ALTER TABLE table_name ENABLE LOGICAL REPLICATION
ALLOWNONVALIDATEDKEYS
```

Mining Mode Toggling

Mining mode toggling is not supported. This implies that after you create mining in `ROOT`, then the session will stay mining in `ROOT` and if the you choose to create mining in PDB, then the session will stay mining in that particular PDB.

However, for a specific need if you need to migrate some previous Oracle GoldenGate session mining in `CDB$ROOT` to mining in specific PDB, then there is a migration process that you can follow:

1. Register a new per-PDB Extract. For example, the SCN returned is `X`.
2. Let the old `ROOT` Extract mine the past `X` (`RECOVERYSCN X`).
3. Stop the old Extract
4. Alter the new Extract so that its current SCN is set to be `Y`.
5. Start the new Extract.

This new Extract picks up from where the old Extract left off.

> **Note:**
>
> There may be some duplicate transactions at SCN `Y`, if there were multiple `txs` committing at the same SCN `Y`. However, Replicat can handle duplicate `txs` by default.

Applying to Pluggable Databases

Replicat can only connect and apply to one pluggable database. To specify the correct one, use a SQL*Net connect string for the database user that you specify with the `USERID` or `USERIDALIAS` parameter. For example: `GGADMIN@DBEAST`.

In the parameter file, specify only the *`schema.object`* in the `TARGET` portion of the `MAP` statements. In the `MAP` portion, identify source objects captured from more than one pluggable database with their three-part names or use the SOURCECATALOG parameter with two-part names. The following is an example of this configuration.

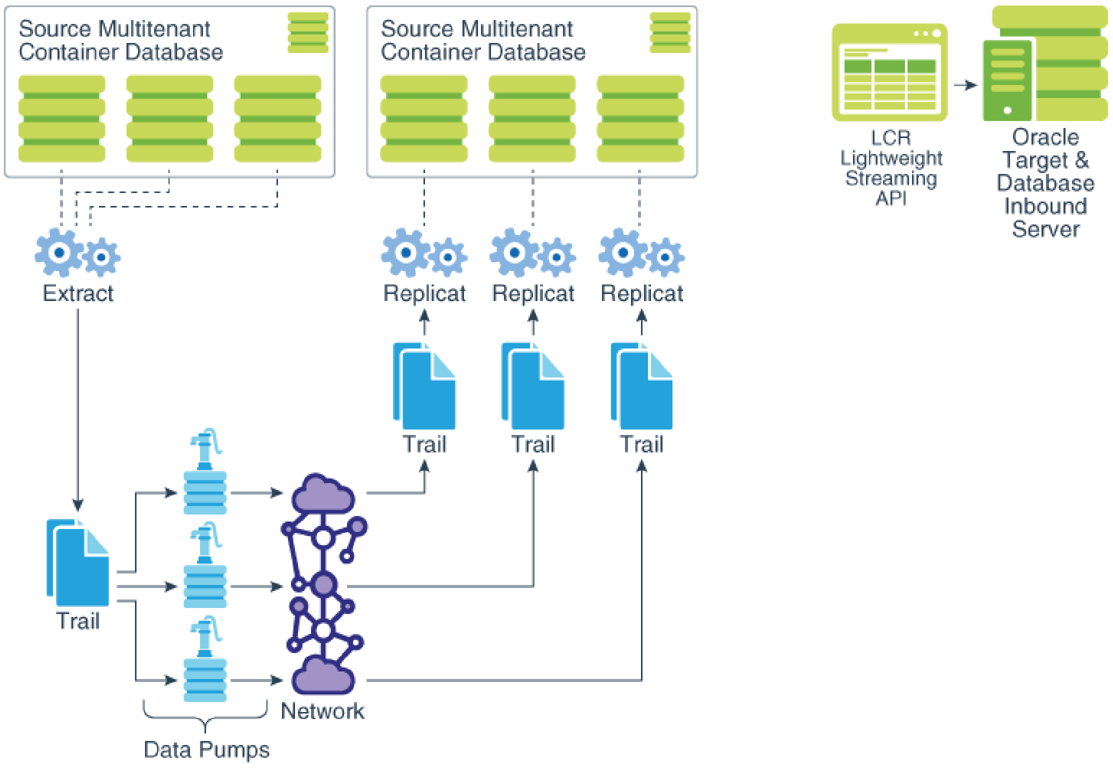
```
SOURCECATALOG pdbeast
MAP hr.employees, TARGET hr.employees;
MAP hr.seq, TARGET hr.employees;
SOURCECATALOG pdbwest
MAP hr.*, TARGET hr.*;
MAP hr.seq, hr.*;
```

The following is an example *without* the use of SOURCECATALOG to identify the source pluggable database. In this case, the source objects are specified with their three-part names.

```
MAP pdbeast.hr.employees, TARGET hr.*;
MAP pdbeast.hr.seq, TARGET hr.*;
```

To configure replication from multiple source pluggable databases to multiple target pluggable databases, you can configure parallel Extract and Replicat streams, each handling data for one pluggable database. Alternatively, you can configure one Extract capturing from multiple source pluggable databases, which writes to one trail that is read by multiple Replicat groups, each applying to a different target pluggable database.

Yet another alternative is to use one Extract writing to multiple trails, each trail read by a Replicat assigned to a specific target pluggable database :


Source Multitenant Container Database
Source Multitenant Container Database
LCR Lightweight Streaming API
Oracle Target & Database Inbound Server
Extract
Replicat
Replicat
Replicat
Trail
Trail
Trail
Trail
Network
Data Pumps

See Configuring Replicat for steps for Replicat configuration at the PDB level and adding Replicat groups.

- Add the Replicat Group
  These steps add the Replicat group that reads the remote trail and applies the data changes to the target Oracle Database.

Add the Replicat Group

These steps add the Replicat group that reads the remote trail and applies the data changes to the target Oracle Database.

1. Connect to the deployment from the Admin Client using the CONNECT command.
2. If using integrated Replicat, issue the `DBLOGIN` command to log into the database.

   ```
   DBLOGIN USERIDALIAS alias
   ```

   Where: *`alias`* specifies the alias of the database login credential that is assigned to Replicat. This credential must exist in the Oracle GoldenGate credential store. For more information, see Establishing Oracle GoldenGate Credentials
3. Issue the `ADD REPLICAT` command with the following syntax.

   ```
   ADD REPLICAT group name, [INTEGRATED,] EXTTRAIL pathname
   ```

   Where:

   - *`group name`* is the name of the Replicat group.
   - `INTEGRATED` creates an integrated Replicat group.
   - `EXTTRAIL` *`pathname`* is the relative or fully qualified name of the remote trail, including the two-character name.

     For more information, see *Reference for Oracle GoldenGate*.

Example 3-3 Adds a Nonintegrated Replicat

```
ADD REPLICAT repe, EXTTRAIL east/ea
```

Example 3-4 Adds an Integrated Replicat

```
ADD REPLICAT repn, INTEGRATED, EXTTRAIL north/em
```

Excluding Objects from the Configuration

To exclude pluggable databases, schemas, and objects from the configuration, you can use the `CATALOGEXCLUDE`, `SCHEMAEXCLUDE`, `TABLEEXCLUDE`, `MAPEXCLUDE`, and `EXCLUDEWILDCARDOBJECTSONLY` parameters.

4

Configuring Oracle GoldenGate Replicat

This chapter contains instructions for configuring the Replicat apply process in either nonintegrated or integrated mode.

Topics:

Choosing from Different Replicat Modes

Learn about the different Replicat modes for your database environment.

Topics:

Deciding Which Apply Method to Use

The Replicat process is responsible for the application of replicated data to an Oracle target database.

For an Oracle target database, you can run Replicat in parallel, non-integrated or integrated mode. Oracle recommends that you use the parallel Replicat unless a specific feature requires a different type of Replicat.

The following table lists the features supported by the respective Replicats.

Feature	Parallel Replicat	Integrated Replicat	Coordinated Replicat	Classic Replicat
Batch Processing	Yes	Yes	Yes	Yes
Barrier Transactions	Yes	Yes	Yes	No
Dependency Computation	Yes	Yes	No	No

Feature	Parallel Replicat	Integrated Replicat	Coordinated Replicat	Classic Replicat
Auto-parallelism	Yes	Yes	No	No

Note: Auto-parallelism is disabled, by default. Only

Feature	Parallel Replicat	Integrated Replicat	Coordinated Replicat	Classic Replicat
	fourthreadsareusedinthedefaultsettings.Ifyouwanttochan			

Feature	Parallel Replicat	Integrated Replicat	Coordinated Replicat	Classic Replicat
	ge Replicat to use `MIN_PARALLELISM` and `MAX_PARALLELISM` then			

Feature	Parallel Replicat	Integrated Replicat	Coordinated Replicat	Classic Replicat
	auto-parallelism is used.			
DML Handler	Yes, Integrated mode	Yes	No	No
Procedural Replication	Yes, used for integrated Parallel Replicat (iPR)	Yes	No	No
Auto CDR	Yes, used by iPR only	Yes	No	No
Dependency-aware Transaction Split	Yes	No	No	No
Cross-RAC-node Processing	Yes	No	Yes	No
`ALLOWDUPTARGETMAP` See `ALLOWDUPTARGETMAP | NOALLOWDUPTARGETMAP`	No. Oracle Database with iPR	No, Oracle Database	Yes	Yes

Topics:

- About Parallel Replicat

- About Non-integrated Replicat
- About Integrated Replicat

About Parallel Replicat

Parallel Replicat is another variant of Replicat that applies transactions in parallel to improve performance.

It takes into account dependencies between transactions, similar to Integrated Replicat. The dependency computation, parallelism of the mapping and apply is performed outside the database so can be off-loaded to another server. The transaction integrity is maintained in this process. In addition, parallel Replicat supports the parallel apply of large transactions by splitting a large transaction into chunks and applying them in parallel.

Note:

For best performance for an OLTP workload, parallel Replicat in non-integrated mode is recommended.

Only Oracle database supports parallel Replicat and integrated parallel Replicat. However, parallel Replicat supports all databases when using the non-integrated option.

To use parallel Replicat, you need to ensure that you have the following values, which are also the default values:

- Metadata in the trail (which means you can't use parallel Replicat if your trails are formatted below 12.1.
- Scheduling columns in the trail file.
- `UPDATERCORDFORMAT COMPACT` parameter.

With integrated parallel Replicat, the Replicat sends the LCRs to the inbound server, which applies the data to the target database, and in regular parallel Replicat, Oracle GoldenGate applies the LCR as a SQL statement directly to the database, similar to how the other non-integrated Replicats work.

The components of parallel Replicat are:

- Mappers operate in parallel to read the trail, map trail records, convert the mapped records to the Integrated Replicat LCR format, and send the LCRs to the Merger for further processing. While one Mapper maps one set of transactions, the next Mapper maps the next set of transactions. The the trail information is split and the trail file is untouched because it orders trail information in order.
- Master processes have two threads, Collater and Scheduler. The Collater receives mapped transactions from the Mappers and puts them back into trail order for dependency calculation. The Scheduler calculates dependencies between transactions, groups transactions into independent batches, and sends the batches to the Appliers to be applied to the target database.
- Appliers reorder records within a batch for array execution. It applies the batch to the target database and performs error handling. It also tracks applied transactions in checkpoint tables.

> **Note:**
>
> Parallel Replicat requires that any foreign key columns are indexed.

- Benefits of Parallel Replicat
- Parallel Replication Architecture
  Parallel replication processes leverage the apply processing functionality that is available within the Oracle Database in integrated mode.

Benefits of Parallel Replicat

The following are the benefits of using parallel Replicat.

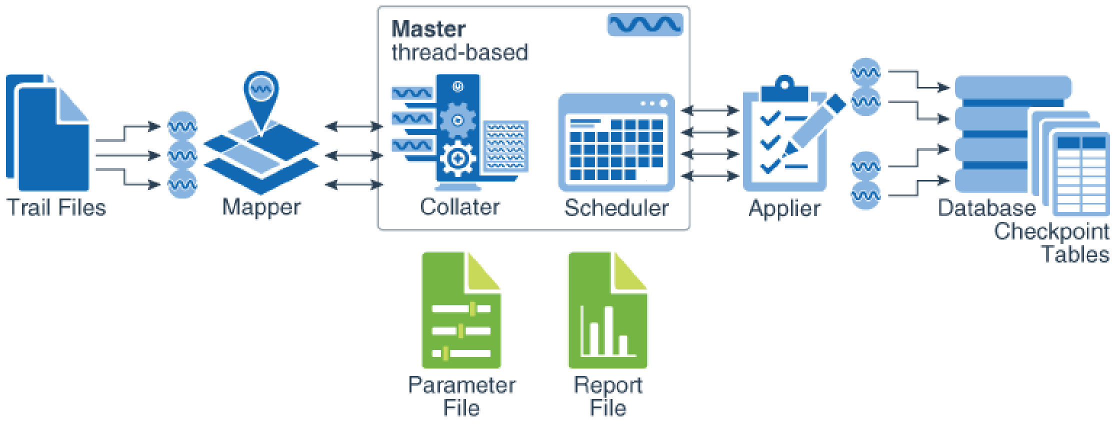
- Integrated Parallel Replicat enables heavy workloads to be partitioned automatically among parallel apply processes that apply multiple transactions concurrently, while preserving the integrity and atomicity of the source transaction. Both a minimum and maximum number of apply processes can be configured with the `PARALLELISM` and `MAX_PARALLELISM` parameters. Replicat automatically adds additional servers when the workload increases, and then adjusts downward again when the workload lightens.
- Integrated Parllel Replicat requires minimal work to configure. All work is configured within one Replicat parameter file, without configuring range partitions.
- High-performance apply streaming is enabled for integrated parallel Replicat by means of a lightweight application programming interface (API) between Replicat and the inbound server.
- Barrier transactions are coordinated by integrated parallel Replicat among multiple server apply processes.
- DDL operations are processed as direct transactions that force a barrier by waiting for server processing to complete before the DDL execution.
- Transient duplicate primary key updates are handled by integrated parallel Replicat in a seamless manner.
- Parallel Replicat can break a single large transaction into smaller chunks and apply those chunks in parallel. See the `SPLIT_TRANS_RECS` for details.

Parallel Replication Architecture

Parallel replication processes leverage the apply processing functionality that is available within the Oracle Database in integrated mode.

Within a single Replicat configuration, multiple inbound server child processes, known as apply servers, apply transactions in parallel while preserving the original transaction atomicity.

The following architecture diagram depicts the flow of change records through the various processes of a parallel replication from the trail files to the target database, for a non-integrated parallel Replicat.



The Mappers read the trail file and map records, forward the mapped records to the Master. The batches are sent to the Appliers where they are applied to the target database.

The Master process consists of two separate threads, Collater and Scheduler. The Collater is responsible for managing and communicating with the Mappers, along with receiving the mapped transactions and reordering them into a single in-order stream. The Scheduler is responsible for managing and communicating with the Appliers, along with reading transactions from the Collater, batching them, and scheduling them to Appliers.

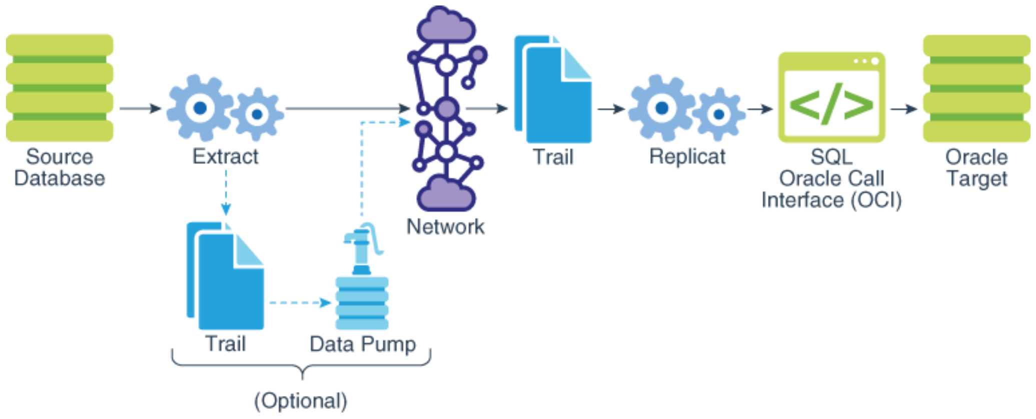
The Scheduler controller communicates with the Scheduler to gather any necessary information (such as, the current low watermark position). The Scheduler controller is required for CDB mode for Oracle Database because it is responsible for aggregating information pertaining to the different target PDBs and reporting a unified picture. The Scheduler controller is created for simplicity and uniformity of implementation, even when not in CDB mode. Every process reads the parameter file and shares a single checkpoint file.

About Non-integrated Replicat

In non-integrated mode, the Replicat process uses standard SQL to apply data directly to the target tables. In this mode, Replicat operates as follows:

- Reads the Oracle GoldenGate trail.
- Performs data filtering, mapping, and conversion.
- Constructs SQL statements that represent source database DML or DDL transactions (in committed order).
- Applies the SQL to the target through Oracle Call Interface (OCI).

The following diagram illustrates the configuration of Replicat in non-integrated mode.



Use non-integrated Replicat when you want to make heavy use of features that are not supported in integrated Replicat mode, see About Integrated Replicat.

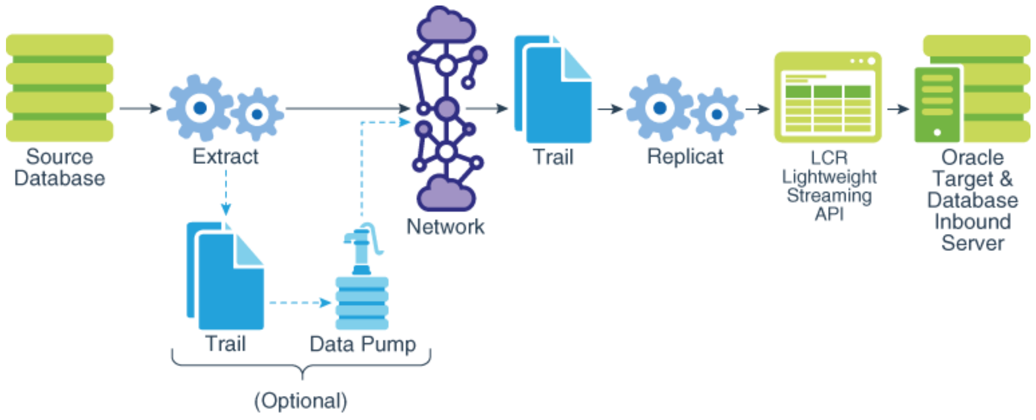
You can apply transactions in parallel with a non-integrated Replicat by using a coordinated Replicat configuration.

About Integrated Replicat

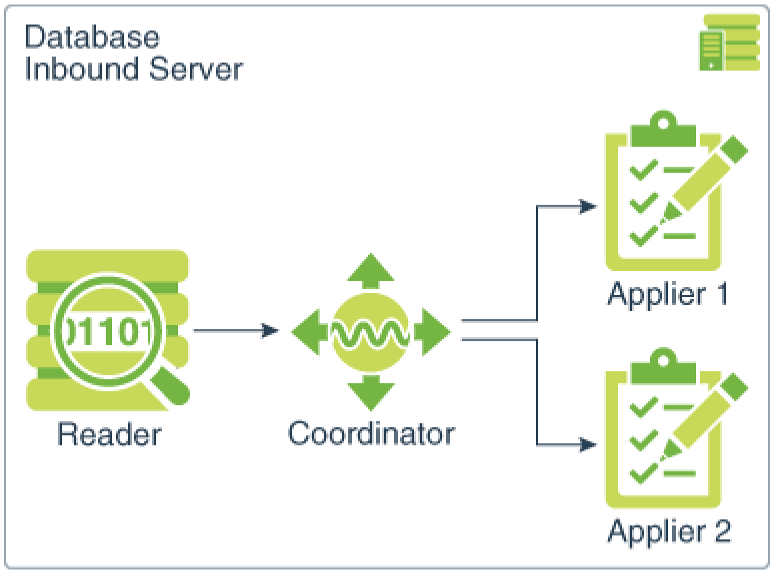
In integrated mode, the Replicat process leverages the apply processing functionality that is available within the Oracle Database. In this mode, Replicat operates as follows:

- Reads the Oracle GoldenGate trail.
- Performs data filtering, mapping, and conversion.
- Constructs logical change records (LCR) that represent source database DML transactions (in committed order). DDL is applied directly by Replicat.
- Attaches to a background process in the target database known as a *database inbound server* by means of a lightweight streaming interface.
- Transmits the LCRs to the inbound server, which applies the data to the target database.

The following figure illustrates the configuration of Replicat in integrated mode.


Source Database
Extract
Network
Trail
Replicat
LCR Lightweight Streaming API
Oracle Target & Database Inbound Server
Trail
Data Pump
(Optional)

Within a single Replicat configuration, multiple inbound server child processes known as *apply servers* apply transactions in parallel while preserving the original transaction atomicity. You can increase this parallelism as much as your target system will support when you configure the Replicat process or dynamically as needed. The following diagram illustrates integrated Replicat configured with two parallel apply servers.



Integrated Replicat applies transactions asynchronously. Transactions that do not have interdependencies can be safely executed and committed out of order to achieve fast throughput. Transactions with dependencies are guaranteed to be applied in the same order as on the source.

A reader process in the inbound server computes the dependencies among the transactions in the workload based on the constraints defined at the target database (primary key, unique, foreign key). Barrier transactions and DDL operations are managed automatically, as well. A coordinator process coordinates multiple transactions and maintains order among the apply servers.

If the inbound server does not support a configured feature or column type, Replicat disengages from the inbound server, waits for the inbound server to complete transactions in its queue, and then applies the transaction to the database in *direct apply* mode through OCI. Replicat resumes processing in integrated mode after applying the direct transaction.

The following features are applied in direct mode by Replicat:

- DDL operations
- Sequence operations
- `SQLEXEC` parameter within a `TABLE` or `MAP` parameter
- `EVENTACTIONS` processing
- `UDT`

> **Note:**
>
> By default, UDT's are applied with the inbound server. Only if `NOUSENATIVEOBJSUPPORT` is in place, then Extract handling is done by Replicat directly.

Because transactions are applied serially in direct apply mode, heavy use of such operations may reduce the performance of the integrated Replicat mode. Integrated Replicat performs best when most of the apply processing can be performed in integrated mode, see Monitoring and Controlling Processing After the Instantiation in *Using Oracle GoldenGate for Oracle Database*.

> **Note:**
>
> User exits are executed in integrated mode. However, user exit may produce unexpected results, if the exit code depends on data in the replication stream.

> **Note:**
>
> Integrated Replicat requires that any foreign key columns are indexed.

- Benefits of Integrated Replicat
- Integrated Replicat Requirements

Benefits of Integrated Replicat

The following are the benefits of using integrated Replicat versus nonintegrated Replicat.

- Integrated Replicat enables heavy workloads to be partitioned automatically among parallel apply processes that apply multiple transactions concurrently, while preserving the integrity and atomicity of the source transaction. Both a minimum and maximum number of apply processes can be configured with the `PARALLELISM` and `MAX_PARALLELISM` parameters. Replicat automatically adds additional servers when the workload increases, and then adjusts downward again when the workload lightens.
- Integrated Replicat requires minimal work to configure. All work is configured within one Replicat parameter file, without configuring range partitions.
- High-performance apply streaming is enabled for integrated Replicat by means of a lightweight application programming interface (API) between Replicat and the inbound server.
- Barrier transactions are coordinated by integrated Replicat among multiple server apply processes.
- DDL operations are processed as direct transactions that force a barrier by waiting for server processing to complete before the DDL execution.

- Transient duplicate primary key updates are handled by integrated Replicat in a seamless manner.

Integrated Replicat Requirements

To use integrated Replicat, the following must be true.

- Supplemental logging must be enabled on the source database to support the computation of dependencies among tables and scheduling of concurrent transactions on the target. Instructions for enabling the required logging are in Configuring Logging Properties. This logging can be enabled at any time up to, but before, you start the Oracle GoldenGate processes.
- Integrated Parallel Replicat is supported on Oracle Database 12.2.0.1 and greater.

Using Different Replicat Modes with Extract

The recommended Oracle GoldenGate configuration, when supported by the Oracle version, is to use one Extract on an Oracle source and one parallel Replicat per source database on an Oracle target.

One integrated Replicat configuration supports all Oracle data types either through the inbound server or by switching to direct apply when necessary, and it preserves source transaction integrity. You can adjust the parallelism settings to the desired apply performance level as needed.

Each Extract group must process objects that are suited to the processing mode, based on table data types and attributes. No objects in one Extract can have DML or DDL dependencies on objects in the other Extract. The same type of segregation must be applied to the Replicat configuration.

If the target database is an Oracle version that does not support integrated Replicat, or if it is a non-Oracle database, you can use a coordinated or parallel Replicat configuration.

Prerequisites for Configuring Replicat

This topic provides the best practices for configuring Replicat.

The guidelines to follow before configuring Replicat are:

1. Preparing the Database for Oracle GoldenGate.
2. Establishing Oracle GoldenGate Credentials.
3. Choosing from Different Replicat Modes.
4. Create the Oracle GoldenGate instance on the target system by configuring the Manager process.

See Before Adding a Replicat in *Step by Step Data Replication Using Oracle GoldenGate Microservices*.

Also see, Quickstart Bidirectional Replication to learn about using the active-active replication process in case of bidirectional replication.

About Checkpoint Table

The checkpoint table is a required component of Replicat.

A Replicat maintains its recovery checkpoints in the checkpoint table, which is stored in the target database. Checkpoints are written to the checkpoint table within the Replicat transaction. Because a checkpoint either succeeds or fails with the transaction, Replicat ensures that a transaction is only applied once, even if there is a failure of the process or the database. See Before Creating Replicat in the *Step by Step Data Replication Using Oracle GoldenGate Microservices* to learn to create checkpoint tables from the Microservices web interface.

> **Note:**
>
> Oracle recommends using checkpoint tables. Multiple classic or coordinated Replicats can share the same checkpoint table, but that may not result in the best performance. With high volume environments, you must ensure that the checkpoint tables do not reside on different drives to become a point of conflict.
>
> See Instantiating Oracle GoldenGate Replication for more information.

- Include the Checkpoint Table in the GLOBALS File
- Disabling Default Asynchronous COMMIT to Checkpoint Table
- Controlling the Checkpoint Retention
  The `CHECKPOINTRETENTIONTIME` option of the `TRANLOGOPTIONS` parameter controls the number of days that Extract retains checkpoints before purging them automatically.

Include the Checkpoint Table in the GLOBALS File

To specify the checkpoint table in the Oracle GoldenGate configuration:

1. Create a `GLOBALS` file (or edit the existing one).

   ```
   EDIT PARAMS ./GLOBALS
   ```

   > **Note:**
   >
   > `EDIT PARAMS` creates a simple text file. When you save the file after `EDIT PARAMS`, it is saved with the name `GLOBALS` in upper case, without a file extension. It must remain as such, and the file must remain in the root Oracle GoldenGate directory.

2. In the `GLOBALS` file, enter the `CHECKPOINTTABLE` parameter.

   ```
   CHECKPOINTTABLE [container.]schema.table
   ```

3. Save and close the `GLOBALS` file.

Disabling Default Asynchronous COMMIT to Checkpoint Table

When a nonintegrated Replicat uses a checkpoint table, it uses an asynchronous `COMMIT` with the `NOWAIT` option to improve performance. Replicat can continue processing immediately after applying this `COMMIT`, while the database logs the transaction in the background. You can disable the asynchronous `COMMIT` with `NOWAIT` by using the `DBOPTIONS` parameter with the `DISABLECOMMITNOWAIT` option in the Replicat parameter file.

> **Note:**
>
> When the configuration of a nonintegrated Replicat group does not include a checkpoint table, the checkpoints are maintained in a file on disk. In this case, Replicat uses `COMMIT` with `WAIT` to prevent inconsistencies in the event of a database failure that causes the state of the transaction, as in the checkpoint file, to be different than its state after the recovery.

Controlling the Checkpoint Retention

The `CHECKPOINTRETENTIONTIME` option of the `TRANLOGOPTIONS` parameter controls the number of days that Extract retains checkpoints before purging them automatically.

Partial days can be specified using decimal values. For example, 8.25 specifies 8 days and 6 hours. The default is seven days.

Configuring Replicat

Configure a Replicat process to configure Replicat for a pluggable database (PDB). Replicat can operate in any of the available modes within an Oracle multitenant container database.

For steps to add a Replicat from the Oracle GoldenGate web interface, see How to Add Replicats. To add a Replicat from the command line interface, see the `ADD REPLICAT`.

Use the following steps to configure the parameters for different Replicat modes.

1. On the target system, create the Replicat parameter file using Oracle GoldenGate Command Line Interface.

   ```
   EDIT PARAMS name
   ```

   Where: *name* is the name of the Replicat group.

2. Enter the Replicat parameters in the order shown, starting a new line for each parameter statement.

   Basic parameters for the Replicat group in nonintegrated mode:

   ```
   REPLICAT repe
   USERIDALIAS ggeast
   ASSUMETARGETDEFS
   MAP hr.*, TARGET hr2.*;
   ```

   Basic parameters for the Replicat group in integrated Replicat mode:

   ```
   REPLICAT repw
   DBOPTIONS INTEGRATEDPARAMS(parallelism 6)
   USERIDALIAS ggwest
   ```

```
ASSUMETARGETDEFS
MAP hr.*, TARGET hr2.*;
```

Parameter	Description
`REPLICAT` *`group`*	*`group`* is the name of the Replicat group.
`DBOPTIONS DEFERREFCONST`	Applies to Replicat in nonintegrated mode. `DEFERREFCONST` sets constraints to `DEFERRABLE` to delay the enforcement of cascade constraints by the target database until the Replicat transaction is committed. See `DBOPTIONS` for additional important information.
`DBOPTIONS INTEGRATEDPARAMS (`*`parameter`*`[, ...])`	This parameter specification applies to Replicat in integrated mode. It specifies optional parameters for the inbound server. See Optional Parameters for Integrated Modesfor additional important information about these `DBOPTIONS` options.
`USERIDALIAS` *`alias`*	Specifies the alias of the database login credential of the user that is assigned to Replicat. This credential must exist in the Oracle GoldenGate credential store. For more information, see Establishing Oracle GoldenGate Credentials
`MAP` *`[container.]schema.object,`* `TARGET` *`schema.object;`*	Specifies the relationship between a source table or sequence, or multiple objects, and the corresponding target object or objects. • `MAP` specifies the source table or sequence, or a wildcarded set of objects. • `TARGET` specifies the target table or sequence or a wildcarded set of objects. • *`container`* is the name of a container, if the source database is a multitenant container database. • *`schema`* is the schema name or a wildcarded set of schemas. • *`object`* is the name of a table or sequence, or a wildcarded set of objects. Terminate this parameter statement with a semi-colon. To exclude objects from a wildcard specification, use the `MAPEXCLUDE` parameter. For more information and for additional options that control data filtering, mapping, and manipulation, see `MAP` in *Reference for Oracle GoldenGate*.

Basic parameters for the Replicat group in parallel Replicat mode:

```
REPLICAT repe
USERID ggadmin, PASSWORD ***
MAP_PARALLELISM 3
MIN_APPLY_PARALLELISM 2
MAX_APPLY_PARALLELISM 10
SPLIT_TRANS_RECS 60000
MAP *.*, TARGET *.*;
```

Parameter	Description
`MAP_PARALLELISM`	Configures number of mappers. This controls the number of threads used to read the trail file. The minimum value is `1`, maximum value is `100` and the default value is `2`.
`APPLY_PARALLELISM`	Configures number of appliers. This controls the number of connections in the target database used to apply the changes. The default value is four.
`MIN_APPLY_PARALLELISM` `MAX_APPLY_PARALLELISM`	The Apply parallelism is auto-tuned. You can set a minimum and maximum value to define the ranges in which the Replicat automatically adjusts its parallelism. There are no defaults. Do *not* use with `APPLY_PARALLELISM` at same time.
`SPLIT_TRANS_REC`	Specifies that large transactions should be broken into pieces of specified size and applied in parallel. Dependencies between pieces are still honored. Disabled by default.
`COMMIT_SERIALIZATION`	Enables commit `FULL` serialization mode, which forces transactions to be committed in trail order.
Advanced Parameters	
`LOOK_AHEAD_TRANSACTIONS`	Controls how far ahead the Scheduler looks when batching transactions. The default value is 10000.
`CHUNK_SIZE`	Controls how large a transaction must be for parallel Replicat to consider it as large. When parallel Replicat encounters a transaction larger than this size, it will serialize it, resulting in decreased performance. However, increasing this value will also increase the amount of memory consumed by parallel Replicat.

3. If using integrated Replicat or parallel Replicat in integrated mode, add the following parameters to the Extract parameter file:
   - `LOGALLSUPCOLS`: This parameter ensures the capture of the supplementally logged columns in the before image. It's the default parameter and shouldn't be turned off or disabled. It is valid for any source database that is supported by Oracle GoldenGate. For Extract versions older than 12c, you can use `GETUPDATEBEFORES` and `NOCOMPRESSDELETES` parameters to satisfy the same requirement. The database must be configured to log the before and after values of the primary key, unique indexes, and foreign keys.
   - The `UPDATERECORDFORMAT` parameter set to COMPACT: This setting causes Extract to combine the before and after images of an `UPDATE` operation into a single record in the trail. This is the default option and it is recommended that you don't change the default setting.
4. Enter any optional Replicat parameters that are recommended for your configuration. You can edit this file at any point before starting processing by using the `EDIT PARAMS` command. See Optional Parameters for Integrated Modes for additional configuration considerations..
5. Save and close the file.

Additional Configuration Steps For Using Nonintegrated Replicat

This chapter contains instructions that are specific only to Replicat when operating in *nonintegrated* mode. When Replicat operates in nonintegrated mode, triggers, cascade constraints, and unique identifiers must be properly configured in an Oracle GoldenGate environment.
This chapter is a supplement to the basic configuration requirements that are documented in Configuring Oracle GoldenGate Replicat.

Topics:

- Disabling Triggers and Referential Cascade Constraints on Target Tables
  Triggers and cascade constraints must be disabled on Oracle target tables when Replicat is in nonintegrated mode.
- Deferring Constraint Checking on Target Tables
  When Replicat is in nonintegrated mode, you may need to defer constraint checking on the target.

Disabling Triggers and Referential Cascade Constraints on Target Tables

Triggers and cascade constraints must be disabled on Oracle target tables when Replicat is in nonintegrated mode.

Oracle GoldenGate provides some options to handle triggers or cascade constraints automatically, depending on the Oracle version:

- For Oracle 11.2.0.2 and later 11gR2 versions, Replicat automatically disables the work performed by triggers during its session. It does not disable a trigger, but instead prevents the trigger body from executing. The `WHEN` portion of the trigger must still compile and execute correctly to avoid database errors. To enable triggers to fire, or to disable them manually, use the `NOSUPPRESSTRIGGERS` option of `DBOPTIONS` and place the statement after the `USERIDALIAS` parameter. To allow a specific trigger to fire, you can use the following database procedure, where *`trigger_owner`* is the owner of the trigger and *`trigger_name`* is the name of the trigger. Once the procedure is called with `FALSE` for a particular trigger, it remains set until the procedure is called with `TRUE`.
- `dbms_ddl.set_trigger_firing_property(trigger_owner "trigger_name", FALSE)`
- For Oracle 11.2.0.2 and later 11gR2 versions, you can use the `DBOPTIONS` parameter with the `DEFERREFCONST` option to delay the checking and enforcement of cascade update and cascade delete constraints until the Replicat transaction commits.
- For other Oracle versions, you must disable triggers and integrity constraints or alter them manually to ignore the Replicat database user.

Constraints must be disabled in nonintegrated Replicat mode because Oracle GoldenGate replicates DML that results from the firing of a trigger or a cascade constraint. If the same trigger or constraint gets activated on the target table, it becomes redundant because of the replicated version, and the database returns an error. Consider the following example, where the source tables are `emp_src` and `salary_src` and the target tables are `emp_targ` and `salary_targ`.

1. A delete is issued for `emp_src`.
2. It cascades a delete to `salary_src`.
3. Oracle GoldenGate sends both deletes to the target.
4. The parent delete arrives first and is applied to `emp_targ`.
5. The parent delete cascades a delete to `salary_targ`.
6. The cascaded delete from `salary_src` is applied to `salary_targ`.
7. The row cannot be located because it was already deleted in step 5.

Deferring Constraint Checking on Target Tables

When Replicat is in nonintegrated mode, you may need to defer constraint checking on the target.

Perform the following steps to defer the constraints:

1. If constraints are `DEFERRABLE` on the source, the constraints on the target must also be `DEFERRABLE`. You can use one of the following parameter statements to defer constraint checking until a Replicat transaction commits:
   - Use `SQLEXEC` at the root level of the Replicat parameter file to defer the constraints for an entire Replicat session.

     ```
     SQLEXEC ("alter session set constraint deferred")
     ```

   - Use the Replicat parameter `DBOPTIONS` with the `DEFERREFCONST` option to delay constraint checking for each Replicat transaction.
2. You might need to configure Replicat to overcome integrity errors caused by transient primary-key duplicates. Transient primary-key duplicates are duplicates that occur temporarily during the execution of a transaction, but are resolved by transaction commit time. This kind of operation typically uses a `SET x = x+n` formula or some other manipulation that shifts values so that a new value equals an existing one.

   The following illustrates a sequence of value changes that can cause a transient primary-key duplicate if constraints are not deferred. The example assumes the primary key column is `CODE` and the current key values (before the updates) are 1, 2, and 3.

   ```
   update item set code = 2 where code = 1;
   update item set code = 3 where code = 2;
   update item set code = 4 where code = 3;
   ```

In this example, when Replicat applies the first update to the target, there is an ORA-00001 (unique constraint) error because the key value of 2 already exists in the table. The Replicat transaction returns constraint violation errors. By default, Replicat does not handle these violations and abends.

- Handling Transient Primary-key Duplicates in Version 11.2.0.4 or Later

Handling Transient Primary-key Duplicates in Version 11.2.0.4 or Later

For versions later than 11.2.0.4, a nonintegrated Replicat by default tries to resolve transient primary-key duplicates automatically by using a workspace in Oracle Workspace Manager. In this configuration, Replicat can defer the constraint checking until commit time without requiring the constraints to be explicitly defined as deferrable.

The requirements for automatic handling of transient primary-key duplicates are:

- Grant the Replicat database user access to the following Oracle function:

  ```
  DBMS_XSTREAM_GG.ENABLE_TDUP_WORKSPACE()
  ```

- The target tables cannot have deferrable constraints; otherwise Replicat returns an error and abends.

To handle tables with deferrable constraints, make certain the constraints are `DEFERRABLE INITIALLY IMMEDIATE`, and use the `HANDLETPKUPDATE` parameter in the `MAP` statement that maps that table. The `HANDLETPKUPDATE` parameter overrides the default of handling the duplicates automatically.The use of a workspace affects the following Oracle GoldenGate error handling parameters:

- `HANDLECOLLISIONS`
- `REPERROR`

When Replicat enables a workspace in Oracle Workspace Manager, it ignores the error handling that is specified by Oracle GoldenGate parameters such as `HANDLECOLLISIONS` and `REPERROR`. Instead, Replicat aborts its grouped transaction (if `BATCHSQL` is enabled), and then retries the update in normal mode by using the active workspace. If ORA-00001 occurs again, Replicat rolls back the transaction and then retries the transaction with error-handling rules in effect again.

Note:

If Replicat encounters ORA-00001 for a non-update record, the error-handling parameters such as `HANDLECOLLISIONS` and `REPERROR` handle it.

A workspace cannot be used if the operation that contains a transient primary-key duplicate also has updates to any out-of-line columns, such as LOB and XMLType. Therefore, these cases are not supported, and any such cases can result in undetected data corruption on the target. An example of this is:

```
update T set PK = PK + 1, C_LOB = 'ABC';
```

Excluding Replicat Transactions

In a bidirectional configuration, Replicat must be configured to mark its transactions, and Extract must be configured to exclude Replicat transactions so that they do not propagate back to their source.

There are two methods to accomplish this as follows:

Method 1

Valid only for Oracle to Oracle implementations.

Replicat can be in either integrated or nonintegrated mode. Use the following parameters:

- Use `DBOPTIONS` with the `SETTAG` option in the Replicat parameter file. The inbound server tags the transactions of that Replicat with the specified value, which identifies those transactions in the redo stream. The default value for `SETTAG` is 00.
- Use the `TRANLOGOPTIONS` parameter with the `EXCLUDETAG` option in an Extract parameter file. The logmining server associated with that Extract excludes redo that is tagged with the `SETTAG` value. Multiple `EXCLUDETAG` statements can be used to exclude different tag values, if desired.

  For Oracle to Oracle, this is the recommended method.

Method 2

Valid for any implementation; Oracle or heterogeneous database configurations.

Alternatively, you could use the Extract `TRANLOGOPTIONS` parameter with the `EXCLUDEUSER` or `EXCLUDEUSERID` option to ignore the Replicat DDL and DML transactions based on its user name or ID. Multiple `EXCLUDEUSER` statements can be used. The specified user is subject to the rules of the `GETREPLICATES` or `IGNOREREPLICATES` parameter.

For more information, see *Reference for Oracle GoldenGate*.

5

Configuring DDL Support

This chapter contains information to help you understand and configure DDL support in Oracle GoldenGate.
Topics:

- Removing Comments from Replicated DDL
  You can use the DDLOPTIONS parameter with the REMOVECOMMENTS BEFORE and REMOVECOMMENTS AFTER options to prevent comments that were used in the source DDL from being included in the target DDL.
- Replicating an IDENTIFIED BY Password
  Use the DDLOPTIONS parameter with the DEFAULTUSERPASSWORDALIAS and REPLICATEPASSWORD | NOREPLICATEPASSWORD options to control how the password of a replicated {CREATE | ALTER} USER *name* IDENTIFIED BY *password* statement is handled. These options must be used together.
- How DDL is Evaluated for Processing
  This topic explains how Oracle GoldenGate processes DDL statements on the source and target systems.
- Viewing DDL Report Information
  By default, Oracle GoldenGate shows basic statistics about DDL at the end of the Extract and Replicat reports.
- Tracing DDL Processing
  If you open a support case with Oracle GoldenGate Technical Support, you might be asked to turn on tracing. TRACE and TRACE2 control DDL tracing.
- Using Edition-Based Redefinition
  Oracle GoldenGate supports the use of Edition-based Redefinition (EBR) with Oracle Databases enabling you to upgrade the database component of an application while it is in use, thereby minimizing or eliminating down time.

Prerequisites for Configuring DDL

Extract can capture DDL operations from a source Oracle database natively through the Oracle logmining server.

- Support for DDL Capture with Extract

Support for DDL Capture with Extract

Extract supports the DDL capture method for **Oracle 11.2.0.4 or later**.

Oracle databases that have the COMPATIBLE parameter set to 11.2.0.4 or higher support DDL capture through the database logmining server. This method is known as *native DDL capture*. Native DDL capture is the only supported method for capturing DDL from a multitenant container database.

For downstream mining, the source database must also have database COMPATIBLE set to 11.2.0.4 or higher to support DDL capture through the database logmining server.

Overview of DDL Synchronization

Oracle GoldenGate supports the synchronization of DDL operations from one database to another.

DDL synchronization can be active when:

- business applications are actively accessing and updating the source and target objects.
- Oracle GoldenGate transactional data synchronization is active.

The components that support the replication of DDL and the replication of transactional data changes (DML) are independent of each other. Therefore, you can synchronize:

- just DDL changes
- just DML changes
- both DDL and DML

For a list of supported objects and operations for DDL support for Oracle, see Supported Objects and Operations in Oracle DDL.

Limitations of Oracle GoldenGate DDL Support

This topic contains some limitations of the DDL feature.

For any additional limitations that were found after this documentation was published, see the *Release Notes for Oracle GoldenGate*.

- DDL Statement Length
- Supported Topologies
- Filtering, Mapping, and Transformation
- Renames
- Interactions Between Fetches from a Table and DDL
- Comments in SQL
- Compilation Errors
- Interval Partitioning
- DML or DDL Performed Inside a DDL Trigger
- LogMiner Data Dictionary Maintenance

DDL Statement Length

Oracle GoldenGate measures the length of a DDL statement in bytes, not in characters. The supported length is approximately 4 MB, allowing for some internal overhead that can vary in size depending on the name of the affected object and its DDL type, among other characteristics. If the DDL is longer than the supported size, Extract will issue a warning and ignore the DDL operation.

Supported Topologies

Oracle GoldenGate supports DDL synchronization only in a like-to-like configuration. The source and target object definitions must be identical.

DDL replication is only supported for Oracle to Oracle replication. It is not supported between different databases, like Oracle to Teradata, or SQL Server to Oracle.

Oracle GoldenGatedoes not support DDL on a standby database.

Oracle GoldenGate supports DDL replication in all supported unidirectional configurations, and in bidirectional configurations between two, and only two, systems. For special considerations in an Oracle active-active configuration, see Propagating DDL in Active-Active (Bidirectional) Configurations.

Filtering, Mapping, and Transformation

DDL operations cannot be transformed by any Oracle GoldenGate process. However, source DDL can be mapped and filtered to a different target object by a primary Extract or a Replicat process. Mapping or filtering of DDL by a data-pump Extract is not permitted, and the DDL is passed as it was received from the primary Extract.

For example, `ALTER TABLE TableA` is processed by a data pump as `ALTER TABLE TableA`. It cannot be mapped by that process as `ALTER TABLE TableB`, regardless of any `TABLE` statements that specify otherwise.

Renames

`RENAME` operations on tables are converted to the equivalent `ALTER TABLE RENAME` so that a schema name can be included in the target DDL statement. For example `RENAME tab1 TO tab2` could be changed to `ALTER TABLE schema.tab1 RENAME TO schema.tab2`. The conversion is reported in the Replicat process report file.

Interactions Between Fetches from a Table and DDL

Oracle GoldenGate supports some data types by identifying the modified row from the redo stream and then querying the underlying table to fetch the changed columns. For instance, partial updates on LOBs are supported by identifying the modified row and the LOB column from the redo log, and then querying for the LOB column value for the row from the base table. A similar technique is employed to support UDT.

Note:

Extract only requires fetch for UDT when *not* using native object support.

Such fetch-based support is implemented by issuing a flashback query to the database based on the SCN (System Change Number) at which the transaction committed. The flashback query feature has certain limitations. Certain DDL operations act as barriers such that flashback queries to get data prior to these DDLs do not succeed. Examples of such DDL are `ALTER TABLE MODIFY COLUMN` and `ALTER TABLE DROP COLUMN`.

Thus, in cases where there is Extract capture lag, an intervening DDL may cause fetch requests for data prior to the DDL to fail. In such cases, Extract falls back and fetches the current snapshot of the data for the modified column. There are several limitations to this approach: First, the DDL could have modified the column that Extract needs to fetch (for example, suppose the intervening DDL added a new attribute to the UDT that is being captured). Second, the DDL could have modified one of the columns that Extract uses as a logical row identifier. Third, the table could have been renamed before Extract had a chance to fetch the data.

To prevent fetch-related inconsistencies such as these, take the following precautions while modifying columns.

1. Pause all DML to the table.
2. Wait for Extract to finish capturing all remaining redo, and wait for Replicat to finish processing the captured data from trail. To determine whether Replicat is finished, issue the following command in GGSCI until you see a message that there is no more data to process.

   ```
   INFO REPLICAT group
   ```

3. Execute the DDL on the source.
4. Resume source DML operations.

Comments in SQL

If a source DDL statement contains a comment in the middle of an object name, that comment will appear at the end of the object name in the target DDL statement. For example:

Source:

```
CREATE TABLE hr./*comment*/emp ...
```

Target:

```
CREATE TABLE hr.emp /*comment*/ ...
```

This does not affect the integrity of DDL synchronization. Comments in any other area of a DDL statement remain in place when replicated.

Compilation Errors

If a `CREATE` operation on a trigger, procedure, function, or package results in compilation errors, Oracle GoldenGate executes the DDL operation on the target anyway. Technically, the DDL operations themselves completed successfully and should be propagated to allow dependencies to be executed on the target, for example in recursive procedures.

Interval Partitioning

DDL replication is unaffected by interval partitioning, because the DDL is implicit. However, this is system generated name so Replicat cannot convert this to the target.I believe this is expected behavior. You must drop the partition on the source. For example:

```
alter table t2 drop partition for (20);
```

DML or DDL Performed Inside a DDL Trigger

DML or DDL operations performed from within a DDL trigger are not captured.

LogMiner Data Dictionary Maintenance

Oracle recommends that you gather dictionary statistics *after* the Extract is registered (logminer session) and the logminer dictionary is loaded, or after any significant DDL activity on the database.

Configuration Guidelines for DDL Support

The following are guidelines to take into account when configuring Oracle GoldenGate processes to support DDL replication.

- Database Privileges
- Parallel Processing
- Object Names
- Data Definitions
- Truncates
- Initial Synchronization
- Data Continuity After CREATE or RENAME

Database Privileges

For database privileges that are required for Oracle GoldenGate to support DDL capture and replication, see Establishing Oracle GoldenGate Credentials .

Parallel Processing

If using parallel Extract and/or Replicat processes, keep related DDL and DML together in the same process stream to ensure data integrity. Configure the processes so that:

- all DDL and DML for any given object are processed by the same Extract group and by the same Replicat group.
- all objects that are relational to one another are processed by the same process group.

For example, if `ReplicatA` processes DML for `Table1`, then it should also process the DDL for `Table1`. If `Table2` has a foreign key to `Table1`, then its DML and DDL operations also should be processed by `ReplicatA`.

If an Extract group writes to multiple trails that are read by different Replicat groups, Extract sends all of the DDL to all of the trails. Use each Replicat group to filter the DDL by using the filter options of the `DDL` parameter in the Replicat parameter file.

Object Names

Oracle GoldenGate preserves the database-defined object name, case, and character set. This support preserves single-byte and multibyte names, symbols, and accent characters at all levels of the database hierarchy.

Object names must be fully qualified with their two-part or three-part names when supplied as input to any parameters that support DDL synchronization. You can use the question mark (**?**) and asterisk (*****) wildcards to specify object names in configuration parameters that support DDL synchronization, but the wildcard specification also must be fully qualified as a two-part or three-part name. To process wildcards correctly, the `WILDCARDRESOLVE` parameter is set to `DYNAMIC` by default. If

`WILDCARDRESOLVE` is set to anything else, the Oracle GoldenGate process that is processing DDL operations will abend and write the error to the process report.

Data Definitions

Because DDL support requires a like-to-like configuration, the `ASSUMETARGETDEFS` parameter must be used in the Replicat parameter file. Replicat will abend if objects are configured for DDL support and the `SOURCEDEFS` parameter is being used. For more information about `ASSUMETARGETDEFS`, see *Reference for Oracle GoldenGate*.

For more information about using a definitions file, see *Administering Oracle GoldenGate*.

Truncates

`TRUNCATE` statements can be supported as follows:

- As part of the Oracle GoldenGate full DDL support, which supports `TRUNCATE TABLE`, `ALTER TABLE TRUNCATE PARTITION`, and other DDL. This is controlled by the DDL parameter (see Enabling DDL Support.)
- As standalone `TRUNCATE` support. This support enables you to replicate `TRUNCATE TABLE`, but no other DDL. The `GETTRUNCATES` parameter controls the standalone `TRUNCATE` feature. For more information, see *Reference for Oracle GoldenGate*.

To avoid errors from duplicate operations, only one of these features can be active at the same time.

Initial Synchronization

To configure DDL replication, start with a target database that is synchronized with the source database. DDL support is compatible with the Replicat initial load method.

Before executing an initial load, disable DDL extraction and replication. DDL processing is controlled by the DDL parameter in the Extract and Replicat parameter files.

After initial synchronization of the source and target data, use all of the source sequence values at least once with `NEXTVAL` before you run the source applications. You can use a script that selects `NEXTVAL` from every sequence in the system. This must be done while Extract is running.

Data Continuity After CREATE or RENAME

To replicate DML operations on new Oracle tables resulting from a `CREATE` or `RENAME` operation, the names of the new tables must be specified in `TABLE` and `MAP` statements in the parameter files. You can use wildcards to make certain that they are included.

To create a new user with `CREATE USER` and then move new or renamed tables into that schema, the new user name must be specified in `TABLE` and `MAP` statements. To create a new user `fin2` and move new or renamed tables into that schema, the parameter statements could look as follows, depending on whether you want the `fin2` objects mapped to the same, or different, schema on the target:

Extract:

```
TABLE fin2.*;
```

Replicat:

```
MAP fin2.*, TARGET different_schema.*;
```

Understanding DDL Scopes

Database objects are classified into scopes. A scope is a category that defines how DDL operations on an object are handled by Oracle GoldenGate.

The scopes are:

- `MAPPED`
- `UNMAPPED`
- `OTHER`

The use of scopes enables granular control over the filtering of DDL operations, string substitutions, and error handling.

- Mapped Scope
- Unmapped Scope
- Other Scope

Mapped Scope

Objects that are specified in `TABLE` and `MAP` statements are of `MAPPED` scope. Extraction and replication instructions in those statements apply to both data (DML) and DDL on the specified objects, unless override rules are applied.

For objects in `TABLE` and `MAP` statements, the DDL operations listed in the following table are supported.

Operations	On any of these Objects[1]
CREATE	TABLE[3]
ALTER	INDEX
DROP	TRIGGER
RENAME	SEQUENCE
COMMENT ON[2]	MATERIALIZED VIEW
	VIEW
	FUNCTION
	PACKAGE
	PROCEDURE
	SYNONYM
	PUBLIC SYNONYM[4]
GRANT	TABLE
REVOKE	SEQUENCE
	MATERIALIZED VIEW

Operations	On any of these Objects[1]
ANALYZE	TABLE INDEX CLUSTER

1 TABLE and MAP do not support some special characters that could be used in an object name affected by these operations. Objects with non-supported special characters are supported by the scopes of UNMAPPED and OTHER.

2 Applies to COMMENT ON TABLE, COMMENT ON COLUMN

3 Includes AS SELECT

4 Table name must be qualified with schema name.

For Extract, MAPPED scope marks an object for DDL capture according to the instructions in the TABLE statement. For Replicat, MAPPED scope marks DDL for replication and maps it to the object specified by the schema and name in the TARGET clause of the MAP statement. To perform this mapping, Replicat issues ALTER SESSION to set the schema of the Replicat session to the schema that is specified in the TARGET clause. If the DDL contains unqualified objects, the schema that is assigned on the target depends on circumstances described in Understanding DDL Scopes.

Assume the following TABLE and MAP statements:

Extract (source)

```
TABLE fin.expen;
TABLE hr.tab*;
```

Replicat (target)

```
MAP fin.expen, TARGET fin2.expen2;
MAP hr.tab*, TARGET hrBackup.bak_*;
```

Also assume a source DDL statement of:

```
ALTER TABLE fin.expen ADD notes varchar2(100);
```

In this example, because the source table fin.expen is in a MAP statement with a TARGET clause that maps to a different schema and table name, the target DDL statement becomes:

```
ALTER TABLE fin2.expen2 ADD notes varchar2(100);
```

Likewise, the following source and target DDL statements are possible for the second set of TABLE and MAP statements in the example:

Source:

```
CREATE TABLE hr.tabPayables ... ;
```

Target:

```
CREATE TABLE hrBackup.bak_tabPayables ...;
```

When objects are of MAPPED scope, you can omit their names from the DDL configuration parameters, unless you want to refine their DDL support further. If you ever need to change the object names in TABLE and MAP statements, the changes will apply automatically to the DDL on those objects.

If you include an object in a `TABLE` statement, but not in a `MAP` statement, the DDL for that object is `MAPPED` in scope on the source but `UNMAPPED` in scope on the target.

Unmapped Scope

If a DDL operation is supported for use in a `TABLE` or `MAP` statement, but its base object name is not included in one of those parameters, it is of `UNMAPPED` scope.

An object name can be of `UNMAPPED` scope on the source (not in an Extract `TABLE` statement), but of `MAPPED` scope on the target (in a Replicat `MAP` statement), or the other way around. When Oracle DDL is of `UNMAPPED` scope in the Replicat configuration, Replicat will by default do the following:

1. Set the current schema of the Replicat session to the schema of the source DDL object.
2. Execute the DDL as that schema.
3. Restore Replicat as the current schema of the Replicat session.

See Understanding DDL Scopes.

Other Scope

DDL operations that cannot be mapped are of `OTHER` scope. When DDL is of `OTHER` scope in the Replicat configuration, it is applied to the target with the same schema and object name as in the source DDL.

An example of `OTHER` scope is a DDL operation that makes a system-specific reference, such as DDL that operates on data file names.

Some other examples of `OTHER` scope:

```
CREATE USER joe IDENTIFIED by joe;
CREATE ROLE ggs_gguser_role IDENTIFIED GLOBALLY;
ALTER TABLESPACE gg_user TABLESPACE GROUP gg_grp_user;
```

See Understanding DDL Scopes.

Correctly Identifying Unqualified Object Names in DDL

Extract captures the current schema (also called session schema) that is in effect when a DDL operation is executed. The current container is also captured if the source is a multitenant container database.

The container and schema are used to resolve unqualified object names in the DDL.

Consider the following example:

```
CONNECT SCOTT/TIGER
CREATE TABLE TAB1 (X NUMBER);
CREATE TABLE SRC1.TAB2(X NUMBER) AS SELECT * FROM TAB1;
```

In both of those DDL statements, the unqualified table `TAB1` is resolved as `SCOTT.TAB1` based on the current schema `SCOTT` that is in effect during the DDL execution.

There is another way of setting the current schema, which is to set the `current_schema` for the session, as in the following example:

```
CONNECT SCOTT/TIGER
ALTER SESSION SET CURRENT_SCHEMA=SRC;
CREATE TABLE TAB1 (X NUMBER);
CREATE TABLE SRC1.TAB2(X NUMBER) AS SELECT * FROM TAB1;
```

In both of those DDL statements, the unqualified table `TAB1` is resolved as `SRC.TAB1` based on the current schema `SRC` that is in effect during the DDL execution.

In both classic and integrated capture modes, Extract captures the current schema that is in effect during DDL execution, and it resolves the unqualified object names (if any) by using the current schema. As a result, `MAP` statements specified for Replicat work correctly for DDL with unqualified object names.

You can also map a source session schema to a different target session schema, if that is required for the DDL to succeed on the target. This mapping is global and overrides any other mappings that involve the same schema names. To map session schemas, use the `DDLOPTIONS` parameter with the `MAPSESSIONSCHEMA` option.

If the default or mapped session schema mapping fails, you can handle the error with the following `DDLERROR` parameter statement, where error 1435 means that the schema does not exist.

```
DDLERROR 1435 IGNORE INCLUDE OPTYPE ALTER OBJTYPE SESSION
```

Enabling DDL Support

Data Definition Language (DDL) is useful in dynamic environments which change constantly.

By default, the status of DDL replication support is as follows:

- On the source, Oracle GoldenGate DDL support is disabled by default. You must configure Extract to capture DDL by using the `DDL` parameter.
- On the target, DDL support is enabled by default, to maintain the integrity of transactional data that is replicated. By default, Replicat will process all DDL operations that the trail contains. If needed, you can use the `DDL` parameter to configure Replicat to ignore or filter DDL operations.

Filtering DDL Replication

By default, all DDL is passed to Extract.

You can use the filtering with DDL parameter method to filter DDL operations so that specific (or all) DDL is applied to the target database according to your requirements. Valid for native DDL capture. This is the preferred method of filtering and is performed within Oracle GoldenGate, and both Extract and Replicat can execute filter criteria. Extract can perform filtering, or it can send the entire DDL to a trail, and then Replicat can perform the filtering. Alternatively, you can filter in a combination of different locations. The `DDL` parameter gives you control over where the filtering is performed, and it also offers more filtering options, including the ability to filter collectively based on the DDL scope (for example, include all `MAPPED` scope).

Note:

If a DDL operation fails in the middle of a `TRANSACTION`, it forces a commit, which means that the transaction spanning the DDL is split into two. The first half is committed and the second half can be restarted. If a recovery occurs, the second half of the transaction cannot be filtered since the information contained in the header of the transaction is no longer there.

- Filtering with the DDL Parameter

Filtering with the DDL Parameter

The `DDL` parameter is the main Oracle GoldenGate parameter for filtering DDL within the Extract and Replicat processes.

When used without options, the `DDL` parameter performs no filtering, and it causes all DDL operations to be propagated as follows:

- As an Extract parameter, it captures all supported DDL operations that are generated on all supported database objects and sends them to the trail.
- As a Replicat parameter, it replicates all DDL operations from the Oracle GoldenGate trail and applies them to the target. This is the same as the default behavior without this parameter.

When used with options, the `DDL` parameter acts as a filtering agent to include or exclude DDL operations based on:

- scope
- object type
- operation type
- object name
- strings in the DDL command syntax or comments, or both

Only one `DDL` parameter can be used in a parameter file, but you can combine multiple inclusion and exclusion options, along with other options, to filter the DDL to the required level.

- `DDL` filtering options are valid for a primary Extract that captures from the transaction source, but not for a data-pump Extract.
- When combined, multiple filter option specifications are linked logically as `AND` statements.
- All filter criteria specified with multiple options must be satisfied for a DDL statement to be replicated.
- When using complex DDL filtering criteria, it is recommended that you test your configuration in a test environment before using it in production.

For `DDL` parameter syntax and additional usage guidelines, see *Reference for Oracle GoldenGate*.

Note:

Before you configure DDL support, it might help to review How DDL is Evaluated for Processing.

Special Filter Cases

This topic describes the special cases that you must consider before creating your DDL filters.

The following are the special cases for creating filter conditions.

- DDL EXCLUDE ALL
- Implicit DDL

DDL EXCLUDE ALL

`DDL EXCLUDE ALL` is a special processing option that is intended primarily for Extract. `DDL EXCLUDE ALL` blocks the replication of DDL operations, but ensures that Oracle GoldenGate continues to keep the object metadata current. When Extract receives DDL directly from the logmining server (triggerless DDL capture mode), current metadata is always maintained.

You can use `DDL EXCLUDE ALL` when using a method other than Oracle GoldenGate to apply DDL to the target and you want Oracle GoldenGate to replicate data changes to the target objects. It provides the current metadata to Oracle GoldenGate as objects change, thus preventing the need to stop and start the Oracle GoldenGate processes. The following special conditions apply to `DDL EXCLUDE ALL`:

- `DDL EXCLUDE ALL` does not require the use of an `INCLUDE` clause.
- When using `DDL EXCLUDE ALL`, you can set the `WILDCARDRESOLVE` parameter to `IMMEDIATE` to allow immediate DML resolution if required.

To prevent all DDL metadata and operations from being replicated, omit the `DDL` parameter entirely.

Implicit DDL

User-generated DDL operations can generate implicit DDL operations. For example, the following statement generates two distinct DDL operations.

```
CREATE TABLE customers (custID number, name varchar2(50), address varchar2(75),
address2 varchar2(75), city varchar2(50), state (varchar2(2), zip number, contact
varchar2(50), areacode number(3), phone number(7), primary key (custID));
```

The first (explicit) DDL operation is the `CREATE TABLE` statement itself.

The second DDL operation is an implicit `CREATE UNIQUE INDEX` statement that creates the index for the primary key. This operation is generated by the database engine, not a user application.

Guidelines for Filtering Implicit DDL

How to filter implicit DDL depends on the mechanism that you are using to filter DDL. See Filtering DDL Replication for more information.

- When the `DDL` parameter is used to filter DDL operations, Oracle GoldenGate filters out any implicit DDL by default, because the explicit DDL will generate the implicit DDL on the target. For example, the target database will create the appropriate index when the `CREATE TABLE` statement in the preceding example is applied by Replicat.
- – If your filtering rules exclude the explicit DDL from being propagated, you must also create a rule to exclude the implicit DDL. For example, if you exclude the `CREATE TABLE` statement in the following example, but do not exclude the implicit `CREATE UNIQUE INDEX` statement, the target database will try to create the index on a non-existent table.

  ```
  CREATE TABLE customers (custID number, name varchar2(50), address
  varchar2(75), address2 varchar2(75), city varchar2(50), state
  (varchar2(2), zip number, contact varchar2(50), areacode number(3),
  phone number(7), primary key (custID));
  ```

  – If your filtering rules permit the propagation of the explicit DDL, you do not need to exclude the implicit DDL. It will be handled correctly by Oracle GoldenGate and the target database.

How Oracle GoldenGate Handles Derived Object Names

DDL operations can contain a *base object* name and also a *derived object* name.

A base object is an object that contains data. A derived object is an object that inherits some attributes of the base object to perform a function related to that object. DDL statements that have both base and derived objects are:

- `RENAME` and `ALTER RENAME`
- `CREATE` and `DROP` on an index, synonym, or trigger

Consider the following DDL statement:

```
CREATE INDEX hr.indexPayrollDate ON TABLE hr.tabPayroll (payDate);
```

In this case, the table is the base object. Its name (`hr.tabPayroll`) is the *base name* and is subject to mapping with `TABLE` or `MAP` under the `MAPPED` scope. The derived object is the index, and its name (`hr.indexPayrollDate`) is the *derived name*.

You can map a derived name in its own `TABLE` or `MAP` statement, separately from that of the base object. Or, you can use one `MAP` statement to handle both. In the case of `MAP`, the conversion of derived object names on the target works as follows.

- MAP Exists for Base Object, But Not Derived Object
- MAP Exists for Base and Derived Objects
- MAP Exists for Derived Object, But Not Base Object
- New Tables as Derived Objects
- Disabling the Mapping of Derived Objects

MAP Exists for Base Object, But Not Derived Object

If there is a `MAP` statement for the base object, but not for the derived object, the result is a schema based on the mapping that matches the derived object name. Derived objects are only mapped if the `MAPDERIVED` option is enabled, which is also the default option.

For example, consider the following:

Extract (source)

```
Table hr.*;
```

Replicat (target)

```
MAP hr.*, TARGET hrBackup.*;
```

Assume the following source DDL statement:

```
CREATE INDEX hr.indexPayrollDate ON TABLE hr.Payroll (payDate);
```

The `CREATE INDEX` statement is executed by Replicat on the target as follows:

```
CREATE INDEX hrBackup.indexPayrollDate ON TABLE hrBackup.Payroll (payDate);
```

In this example, the mapping is such that it matches the derived object name because of which the derived object schema is changed from `hr` to `hrBackup`.

Here's another example, where there is no mapping that matches the derived object name so the derived object name remains the same.

Extract (source)

```
Table hr.tab*;
```

Replicat (target)

```
MAP hr.tab*, TARGET hrBackup.*;
```

Assume the following source DDL statement:

```
CREATE INDEX hr.indexPayrollDate ON TABLE hr.tabPayroll (payDate);
```

The `CREATE INDEX` statement is executed by Replicat on the target as follows:

```
CREATE INDEX hr.indexPayrollDate ON TABLE hrBackup.tabPayroll (payDate);
```

MAP Exists for Base and Derived Objects

If there is a `MAP` statement for the base object and also one for the derived object, the result is an explicit mapping. Assuming the DDL statement includes `MAPPED`, Replicat converts the schema and name of each object according to its own `TARGET` clause. For example, assume the following:

Extract (source)

```
TABLE hr.tab*;  TABLE hr.index*;
```

Replicat (target)

```
MAP hr.tab*, TARGET hrBackup.*;MAP hr.index*, TARGET hrIndex.*;
```

Assume the following source DDL statement:

```
CREATE INDEX hr.indexPayrollDate ON TABLE hr.tabPayroll (payDate);
```

The `CREATE INDEX` statement is executed by Replicat on the target as follows:

```
CREATE INDEX hrIndex.indexPayrollDate ON TABLE hrBackup.tabPayroll (payDate);
```

Use an explicit mapping when the index on the target must be owned by a different schema from that of the base object, or when the name on the target must be different from that of the source.

MAP Exists for Derived Object, But Not Base Object

If there is a `MAP` statement for the derived object, but not for the base object, Replicat does not perform any name conversion for either object. The target DDL statement is the same as that of the source. To map a derived object, the choices are:

- Use an explicit `MAP` statement for the base object.
- If names permit, map both base and derived objects in the same `MAP` statement by means of a wildcard.
- Create a `MAP` statement for each object, depending on how you want the names converted.

New Tables as Derived Objects

The following explains how Oracle GoldenGate handles new tables that are created from:

- `RENAME` and `ALTER RENAME`
- `CREATE TABLE AS SELECT`
- CREATE TABLE AS SELECT
- RENAME and ALTER TABLE RENAME

CREATE TABLE AS SELECT

The `CREATE TABLE AS SELECT` (CTAS) statements include `SELECT` statements and `INSERT` statements that reference any number of underlying objects. By default, Oracle GoldenGate obtains the data for the `AS SELECT` clause from the target database. You can force the CTAS operation to preserve the original inserts using this parameter.

Note:

For this reason, Oracle `XMLType` tables created from a `CTAS` (`CREATE TABLE AS SELECT`) statement cannot be supported. For `XMLType` tables, the row object IDs must match between source and target, which cannot be maintained in this scenario. `XMLType` tables created by an empty `CTAS` statement (that does not insert data in the new table) can be maintained correctly.

In addition, you could use the `GETCTASDML` parameter that allows CTAS to replay the inserts of the CTAS thus preserving OIDs during replication. This parameter is only supported with Integrated Dictionary and any downstream Replicat must be 12.1.2.1 or greater to consume the trail otherwise, there may be divergence.

The objects in the `AS SELECT` clause must exist in the target database, and their names must be identical to the ones on the source.

In a `MAP` statement, Oracle GoldenGate only maps the name of the new table (`CREATE TABLE name`) to the `TARGET` specification, but does not map the names of the underlying objects from the `AS SELECT` clause. There could be dependencies on those objects that could cause data inconsistencies if the names were converted to the `TARGET` specification.

The following shows an example of a `CREATE TABLE AS SELECT` statement on the source and how it would be replicated to the target by Oracle GoldenGate.

```
CREATE TABLE a.tab1 AS SELECT * FROM a.tab2;
```

The `MAP` statement for Replicat is as follows:

```
MAP a.tab*, TARGET a.x*;
```

The target DDL statement that is applied by Replicat is the following:

```
CREATE TABLE a.xtab1 AS SELECT * FROM a.tab2;
```

The name of the table in the `AS SELECT * FROM` clause remains as it was on the source: `tab2` (rather than `xtab2`).

To keep the data in the underlying objects consistent on source and target, you can configure them for data replication by Oracle GoldenGate. In the preceding example, you could use the following statements to accommodate this requirement:

Source

```
TABLE a.tab*;
```

Target

```
MAPEXCLUDE a.tab2
MAP a.tab*, TARGET a.x*;
MAP a.tab2, TARGET a.tab2;
```

See Correctly Identifying Unqualified Object Names in DDL.

RENAME and ALTER TABLE RENAME

In `RENAME` and `ALTER TABLE RENAME` operations, the base object is always the new table name. In the following example, the base object name is considered to be `index_paydate`.

```
ALTER TABLE hr.indexPayrollDate RENAME TO index_paydate;
```

or...

```
RENAME hr.indexPayrollDate TO index_paydate;
```

The derived object name is `hr.indexPayrollDate`.

Disabling the Mapping of Derived Objects

Use the `DDLOPTIONS` parameter with the `NOMAPDERIVED` option to prevent the conversion of the name of a derived object according to a `TARGET` clause of a `MAP` statement that includes it. `NOMAPDERIVED` overrides any explicit `MAP` statements that contain the name of the base or derived object. Source DDL that contains derived objects is replicated to the target with the same schema and object names as on the source.

The following table shows the results of `MAPDERIVED` compared to `NOMAPDERIVED`, based on whether there is a `MAP` statement just for the base object, just for the derived object, or for both.

Base Object	Derived Object	MAP/NOMAP DERIVED?	Derived object converted per a MAP?	Derived object gets schema of base object?
mapped[1]	mapped	`MAPDERIVED`	yes	no
mapped	not mapped	`MAPDERIVED`	no	yes
not mapped	mapped	`MAPDERIVED`	no	no
not mapped	not mapped	`MAPDERIVED`	no	no
mapped	mapped	`NOMAPDERIVED`	no	no
mapped	not mapped	`NOMAPDERIVED`	no	no
not mapped	mapped	`NOMAPDERIVED`	no	no
not mapped	not mapped	`NOMAPDERIVED`	no	no

1 Mapped means included in a `MAP` statement.

The following examples illustrate the results of `MAPDERIVED` as compared to `NOMAPDERIVED`. In the following table, both trigger and table are owned by `rpt` on the target because both base and derived names are converted by means of `MAPDERIVED`.

MAP statement	Source DDL statement captured by Extract	Target DDL statement applied by Replicat
`MAP fin.*, TARGET rpt.*;`	`CREATE TRIGGER fin.act_trig ON fin.acct;`	`CREATE TRIGGER rpt.act_trig ON rpt.acct;`

In the following table, the trigger is owned by `fin`, because conversion is prevented by means of `NOMAPDERIVED`.

MAP statement	Source DDL statement captured by Extract	Target DDL statement applied by Replicat
`MAP fin.*, TARGET rpt.*;`	`CREATE TRIGGER fin.act_trig ON fin.acct;`	`CREATE TRIGGER fin.act_trig ON rpt.acct;`

Note:

In the case of a `RENAME` statement, the new table name is considered to be the base table name, and the old table name is considered to be the derived table name.

Using DDL String Substitution

You can substitute strings within a DDL operation while it is being processed by Oracle GoldenGate.

This feature provides a convenience for changing and mapping directory names, comments, and other things that are not directly related to data structures. For example, you could substitute one tablespace name for another, or substitute a string within comments. String substitution is controlled by the `DDLSUBST` parameter. For more information, see *Reference for Oracle GoldenGate*.

Note:

Before you create a `DDLSUBST` parameter statement, it might help to review How DDL is Evaluated for Processing in this chapter.

Controlling the Propagation of DDL to Support Different Topologies

To support bidirectional and cascading replication configurations, it is important for Extract to be able to identify the DDL that is performed by Oracle GoldenGate and by other applications, such as the local business applications.

Depending on the configuration that you want to deploy, it might be appropriate to capture one or both of these sources of DDL on the local system.

Note:

Oracle GoldenGate DDL consists of `ALTER TABLE` statements performed by Extract to create log groups and the DDL that is performed by Replicat to replicate source DDL changes.

The following options of the `DDLOPTIONS` parameter control whether DDL on the local system is captured by Extract and then sent to a remote system, assuming Oracle GoldenGate DDL support is configured and enabled:

- The `GETREPLICATES` and `IGNOREREPLICATES` options control whether Extract captures or ignores the DDL that is generated by Oracle GoldenGate. The default is `IGNOREREPLICATES`, which does not propagate the DDL that is generated by Oracle GoldenGate. To identify the DDL operations that are performed by Oracle GoldenGate, the following comment is part of each Extract and Replicat DDL statement:

  ```
  /* GOLDENGATE_DDL_REPLICATION */
  ```

- The `GETAPPLOPS` and `IGNOREAPPLOPS` options control whether Extract captures or ignores the DDL that is generated by applications other than Oracle GoldenGate. The default is `GETAPPLOPS`, which propagates the DDL from local applications (other than Oracle GoldenGate).

The result of these default settings is that Extract ignores its own DDL and the DDL that is applied to the local database by a local Replicat, so that the DDL is not sent back to its source, and Extract captures all other DDL that is configured for replication. The following is the default `DDLOPTIONS` configuration.

```
DDLOPTIONS GETAPPLOPS, IGNOREREPLICATES
```

This behavior can be modified. See the following topics:

- Propagating DDL in Active-Active (Bidirectional) Configurations
- Propagating DDL in a Cascading Configuration

Propagating DDL in Active-Active (Bidirectional) Configurations

Oracle GoldenGate supports active-active DDL replication between two systems. For an active-active bidirectional replication, the following must be configured in the Oracle GoldenGate processes:

1. DDL that is performed by a business application on one system must be replicated to the other system to maintain synchronization. To satisfy this requirement, include the `GETAPPLOPS` option in the `DDLOPTIONS` statement in the Extract parameter files on both systems.
2. DDL that is applied by Replicat on one system must be captured by the local Extract and sent back to the other system. To satisfy this requirement, use the `GETREPLICATES` option in the `DDLOPTIONS` statement in the Extract parameter files on both systems.

> **Note:**
>
> An internal Oracle GoldenGate token will cause the actual Replicat DDL statement itself to be ignored to prevent loopback. The purpose of propagating Replicat DDL back to the original system is so that the Replicat on that system can update its object metadata cache, in preparation to receive incoming DML, which will have the new metadata.

3. Each Replicat must be configured to update its object metadata cache whenever the remote Extract sends over a captured Replicat DDL statement. To satisfy this requirement, use the `UPDATEMETADATA` option in the `DDLOPTIONS` statement in the Replicat parameter files on both systems.

The resultant `DDLOPTIONS` statements should look as follows:

Extract (primary and secondary)

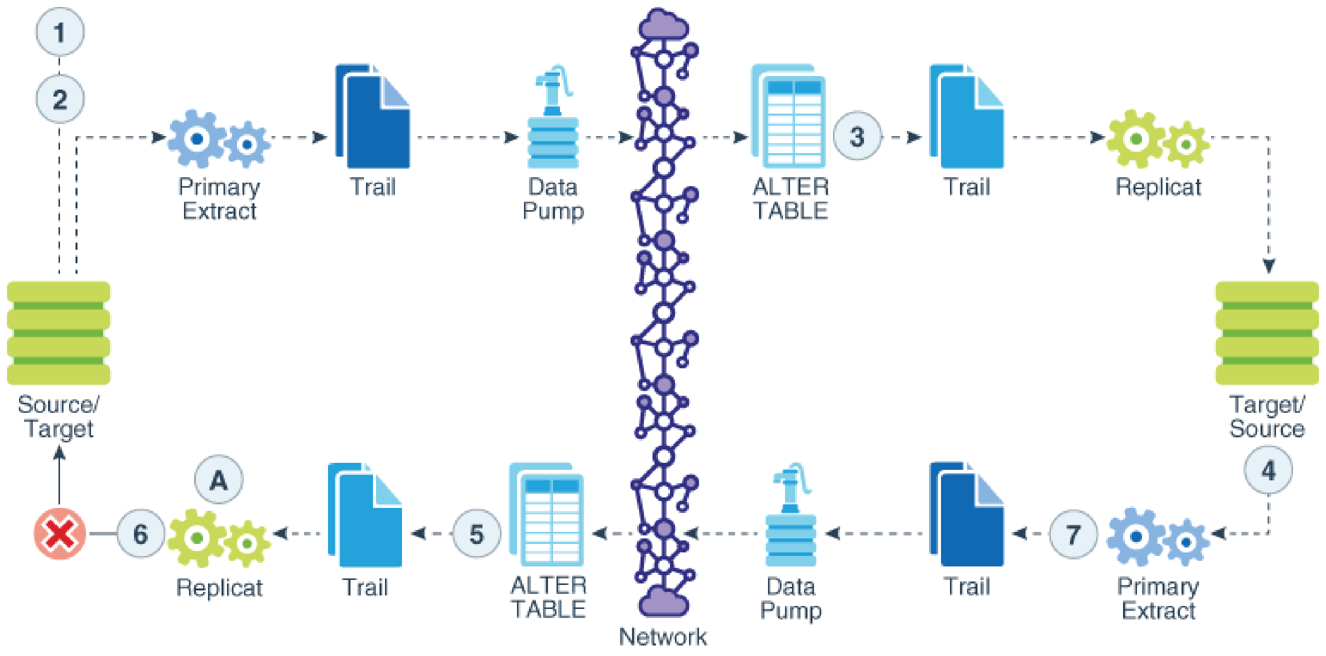
```
DDLOPTIONS GETREPLICATES, GETAPPLOPS
```

Replicat (primary and secondary)

```
DDLOPTIONS UPDATEMETADATA
```

WARNING:

Before you allow new DDL or DML to be issued for the same object(s) as the original DDL, allow time for the original DDL to be replicated to the remote system and then captured again by the Extract on that system. This will ensure that the operations arrive in correct order to the Replicat on the original system, to prevent DML errors caused by metadata inconsistencies. See the following diagram for more information.



The labels in the diagrams imply the following:

- 1: ALTER TABLE Customer ADD Birth_Date date
- 2; New metadata: First_Name varchar2(50), Last_Name varchar2(50), Address varchar2(50), City varchar2(50), Country varchar2(25), Birth_Date date
- 3: ALTER TABLE

- 4: New metadata: First_Name varchar2(50), Last_Name varchar2(50), Address varchar2(50), City varchar2(50), Country varchar2(25), Birth_Date date
- 5: ALTER TABLE
- 6: DDLOPTIONS UPDATEMETADATA New metadata: First_Name varchar2(50), Last_Name varchar2(50), Address varchar2(50), City varchar2(50), Country varchar2(25), Birth_Date date

For more information about `DDLOPTIONS`, see *Reference for Oracle GoldenGate*.

For more information about configuring a bidirectional configuration, see *Administering Oracle GoldenGate*.

Propagating DDL in a Cascading Configuration

In a cascading configuration, use the following setting for `DDLOPTIONS` in the Extract parameter file on each intermediary system. This configuration forces Extract to capture the DDL from Replicat on an intermediary system and cascade it to the next system downstream.

```
DDLOPTIONS GETREPLICATES, IGNOREAPPLOPS
```

For more information about `DDLOPTIONS`, see `DDLOPTIONS` in *Reference for Oracle GoldenGate*.

Adding Supplemental Log Groups Automatically

Use the `DDLOPTIONS` parameter with the `ADDTRANDATA` option for performing tasks described in this topic.

You can perform the following tasks using the `DDLOPTIONS`:

- Enable Oracle's supplemental logging automatically for new tables created with a `CREATE TABLE`.
- Update Oracle's supplemental logging for tables affected by an `ALTER TABLE` to add or drop columns.
- Update Oracle's supplemental logging for tables that are renamed.
- Update Oracle's supplemental logging for tables where unique or primary keys are added or dropped.

To use `DDLOPTIONS ADDSCHEMATRANDATA`, the `ADD SCHEMATRANDATA` command must be issued in GGSCI to enable schema-level supplemental logging.

By default, the `ALTER TABLE` that adds the supplemental logging is not replicated to the target unless the `GETREPLICATES` parameter is in use.

`DDLOPTIONS ADDTRANDATA` is not supported for multitenant container databases, see Configuring Logging Properties for more information.

Removing Comments from Replicated DDL

You can use the `DDLOPTIONS` parameter with the `REMOVECOMMENTS BEFORE` and `REMOVECOMMENTS AFTER` options to prevent comments that were used in the source DDL from being included in the target DDL.

By default, comments are not removed, so that they can be used for string substitution.

Replicating an IDENTIFIED BY Password

Use the `DDLOPTIONS` parameter with the `DEFAULTUSERPASSWORDALIAS` and `REPLICATEPASSWORD | NOREPLICATEPASSWORD` options to control how the password of a replicated `{CREATE | ALTER} USER` *`name`* `IDENTIFIED BY` *`password`* statement is handled. These options must be used together.

See the `USEPASSWORDVERIFIERLEVEL` option of `DDLOPTIONS` for important information about specifying the password verifier when Replicat operates against an Oracle 10*g* or 11*g* database.

Note:

Replication of `CREATE | ALTER PROFILE` will fail as the profile/password verification function must exist in the SYS schema. To replicate these DDLs successfully, password verification function must be created manually on both source/target(s) since DDL to SYS schema is excluded.

How DDL is Evaluated for Processing

This topic explains how Oracle GoldenGate processes DDL statements on the source and target systems.

It shows the order in which different criteria in the Oracle GoldenGate parameters are processed, and it explains the differences between how Extract and Replicat each process the DDL.

Extract

1. Extract captures a DDL statement.
2. Extract separates comments, if any, from the main statement.
3. Extract searches for the `DDL` parameter. (This example assumes it exists.)
4. Extract searches for the `IGNOREREPLICATES` parameter. If it is present, and if Replicat produced this DDL on this system, Extract ignores the DDL statement. (This example assumes no Replicat operations on this system.)
5. Extract determines whether the DDL statement is a `RENAME`. If so, the rename is flagged internally.
6. Extract gets the base object name and, if present, the derived object name.
7. If the statement is a `RENAME`, Extract changes it to `ALTER TABLE RENAME`.

8. Extract searches for the `DDLOPTIONS REMOVECOMMENTS BEFORE` parameter. If it is present, Extract removes the comments from the DDL statement, but stores them in case there is a `DDL INCLUDE` or `DDL EXCLUDE` clause that uses `INSTR` or `INSTRCOMMENTS`.
9. Extract determines the DDL scope: `MAPPED`, `UNMAPPED` or `OTHER`:
   - It is `MAPPED` if the operation and object types are supported for mapping, and the base object name and/or derived object name (if `RENAME`) is in a `TABLE` parameter.
   - It is `UNMAPPED` if the operation and object types are not supported for mapping, and the base object name and/or derived object name (if `RENAME`) is not in a `TABLE` parameter.
   - Otherwise the operation is identified as `OTHER`.
10. Extract checks the `DDL` parameter for `INCLUDE` and `EXCLUDE` clauses, and it evaluates the `DDL` parameter criteria in those clauses. All options must evaluate to `TRUE` in order for the `INCLUDE` or `EXCLUDE` to evaluate to `TRUE`. The following occurs:
    - If an `EXCLUDE` clause evaluates to `TRUE`, Extract discards the DDL statement and evaluates another DDL statement. In this case, the processing steps start over.
    - If an `INCLUDE` clause evaluates to `TRUE`, or if the `DDL` parameter does not have any `INCLUDE` or `EXCLUDE` clauses, Extract includes the DDL statement, and the processing logic continues.
11. Extract searches for a `DDLSUBST` parameter and evaluates the `INCLUDE` and `EXCLUDE` clauses. If the criteria in those clauses add up to `TRUE`, Extract performs string substitution. Extract evaluates the DDL statement against each `DDLSUBST` parameter in the parameter file. For all true `DDLSUBST` specifications, Extract performs string substitution in the order that the `DDLSUBST` parameters are listed in the file.
12. Now that `DDLSUBT` has been processed, Extract searches for the `REMOVECOMMENTS AFTER` parameter. If it is present, Extract removes the comments from the DDL statement.
13. Extract searches for `DDLOPTIONS ADDTRANDATA`. If it is present, and if the operation is `CREATE TABLE`, Extract issues the `ALTER TABLE` *`name`* `ADD SUPPLEMENTAL LOG GROUP` command on the table.
14. Extract writes the DDL statement to the trail.

Replicat

1. Replicat reads the DDL statement from the trail.
2. Replicat separates comments, if any, from the main statement.
3. Replicat searches for `DDLOPTIONS REMOVECOMMENTS BEFORE`. If it is present, Replicat removes the comments from the DDL statement.
4. Replicat evaluates the DDL synchronization scope to determine if the DDL qualifies for name mapping. Anything else is of `OTHER` scope.
5. Replicat evaluates the `MAP` statements in the parameter file. If the source base object name for this DDL (as read from the trail) appears in any of the `MAP`

statements, the operation is marked as `MAPPED` in scope. Otherwise it is marked as `UNMAPPED` in scope.

6. Replicat replaces the source base object name with the base object name that is specified in the `TARGET` clause of the `MAP` statement.
7. If there is a derived object, Replicat searches for `DDLOPTIONS MAPDERIVED`. If it is present, Replicat replaces the source derived name with the target derived name from the `MAP` statement.
8. Replicat checks the `DDL` parameter for `INCLUDE` and `EXCLUDE` clauses, and it evaluates the `DDL` parameter criteria contained in them. All options must evaluate to `TRUE` in order for the `INCLUDE` or `EXCLUDE` to evaluate to `TRUE`. The following occurs:
   - If any `EXCLUDE` clause evaluates to `TRUE`, Replicat discards the DDL statement and starts evaluating another DDL statement. In this case, the processing steps start over.
   - If any `INCLUDE` clause evaluates to `TRUE`, or if the `DDL` parameter does not have any `INCLUDE` or `EXCLUDE` clauses, Replicat includes the DDL statement, and the processing logic continues.
9. Replicat searches for the `DDLSUBST` parameter and evaluates the `INCLUDE` and `EXCLUDE` clauses. If the options in those clauses add up to `TRUE`, Replicat performs string substitution. Replicat evaluates the DDL statement against each `DDLSUBST` parameter in the parameter file. For all true `DDLSUBST` specifications, Replicat performs string substitution in the order that the `DDLSUBST` parameters are listed in the file.
10. Now that `DDLSUBT` has been processed, Replicat searches for the `REMOVECOMMENTS AFTER` parameter. If it is present, Replicat removes the comments from the DDL statement.
11. Replicat executes the DDL statement on the target database.
12. If there are no errors, Replicat processes the next DDL statement. If there are errors, Replicat performs the following steps.
13. Replicat analyzes the `INCLUDE` and `EXCLUDE` rules in the Replicat `DDLERROR` parameters in the order that they appear in the parameter file. If Replicat finds a rule for the error code, it applies the specified error handling; otherwise, it applies `DEFAULT` handling.
14. If the error handling does not enable the DDL statement to succeed, Replicat does one of the following: abends, ignores the operation, or discards it as specified in the rules.

Note:

If there are multiple targets for the same source in a `MAP` statement, the processing logic executes for each one.

Viewing DDL Report Information

By default, Oracle GoldenGate shows basic statistics about DDL at the end of the Extract and Replicat reports.

To enable expanded DDL reporting, use the `DDLOPTIONS` parameter with the `REPORT` option. Expanded reporting includes the following information about DDL processing:

- A step-by-step history of the DDL operations that were processed by Oracle GoldenGate.
- The DDL filtering and processing parameters that are being used.

Expanded DDL report information increases the size of the report file, but it might be useful in certain situations, such as for troubleshooting or to determine when an `ADD TRANDATA` to add supplemental logging was applied.

To view a report, use the `VIEW REPORT` command.

```
VIEW REPORT group
```

- Viewing DDL Reporting in Replicat
- Viewing DDL Reporting in Extract
- Statistics in the Process Reports

Viewing DDL Reporting in Replicat

The Replicat report lists:

- The entire syntax and source Oracle GoldenGate SCN of each DDL operation that Replicat processed from the trail. You can use the source SCN for tracking purposes, especially when there are restores from backup and Replicat is positioned backward in the trail.
- A subsequent entry that shows the scope of the operation (`MAPPED`, `UNMAPPED`, `OTHER`) and how object names were mapped in the target DDL statement, if applicable.
- Another entry that shows how processing criteria was applied.
- Additional entries that show whether the operation succeeded or failed, and whether or not Replicat applied error handling rules.

The following excerpt from a Replicat report illustrates a sequence of steps, including error handling:

```
2011-01-20 15:11:45  GGS INFO     2104  DDL found, operation [drop table
myTableTemp ], Source SCN [1186713.0].
 2011-01-20 15:11:45  GGS INFO     2100  DDL is of mapped scope, after mapping
new operation [drop table "QATEST2"."MYTABLETEMP" ].
 2011-01-20 15:11:45  GGS INFO     2100  DDL operation included [include objname
myTable*], optype [DROP], objtype [TABLE], objname [QATEST2.MYTABLETEMP].
 2011-01-20 15:11:45  GGS INFO     2100  Executing DDL operation.
 2011-01-20 15:11:48  GGS INFO     2105  DDL error ignored for next retry: error
code [942], filter [include objname myTableTemp], error text [ORA-00942: table
or view does not exist], retry [1].
 2011-01-20 15:11:48  GGS INFO     2100  Executing DDL operation , trying again
due to RETRYOP parameter.
 2011-01-20 15:11:51  GGS INFO     2105  DDL error ignored for next retry: error
code [942], filter [include objname myTableTemp], error text [ORA-00942: table
or view does not exist], retry [2].
 2011-01-20 15:11:51  GGS INFO     2100  Executing DDL operation, trying again
due to RETRYOP parameter.
 2011-01-20 15:11:54  GGS INFO     2105  DDL error ignored for next retry: error
code [942], filter [include objname myTableTemp], error text [ORA-00942: table
or view does not exist], retry [3].
 2011-01-20 15:11:54  GGS INFO     2100  Executing DDL operation, trying again
```

```
due to RETRYOP parameter.
 2011-01-20 15:11:54  GGS INFO     2105  DDL error ignored: error code [942], filter
[include objname myTableTemp], error text [ORA-00942: table or view does not exist].
```

Viewing DDL Reporting in Extract

The Extract report lists the following:

- The entire syntax of each captured DDL operation, the start and end SCN, the Oracle instance, the DDL sequence number (from the `SEQNO` column of the history table), and the size of the operation in bytes.
- A subsequent entry that shows how processing criteria was applied to the operation, for example string substitution or `INCLUDE` and `EXCLUDE` filtering.
- Another entry showing whether the operation was written to the trail or excluded.

The following, taken from an Extract report, shows an included operation and an excluded operation. There is a report message for the included operation, but not for the excluded one.

```
2011-01-20 15:11:41  GGS INFO     2100  DDL found, operation [create table myTable (
    myId number (10) not null,
    myNumber number,
    myString varchar2(100),
    myDate date,
    primary key (myId)
) ], start SCN [1186754], commit SCN [1186772] instance [test11g (1)], DDL seqno
[4134].

2011-01-20 15:11:41  GGS INFO     2100  DDL operation included [INCLUDE OBJNAME
myTable*], optype [CREATE], objtype [TABLE], objname [QATEST1.MYTABLE].

2011-01-20 15:11:41  GGS INFO     2100  DDL operation written to extract trail file.

2011-01-20 15:11:42  GGS INFO     2100  Successfully added TRAN DATA for table with
the key, table [QATEST1.MYTABLE], operation [ALTER TABLE "QATEST1"."MYTABLE" ADD
SUPPLEMENTAL LOG GROUP "GGS_MYTABLE_53475" (MYID) ALWAYS  /*
GOLDENGATE_DDL_REPLICATION */ ].

2011-01-20 15:11:43  GGS INFO     2100  DDL found, operation [create table myTableTemp
(
    vid varchar2(100),
    someDate date,
    primary key (vid)
) ], start SCN [1186777], commit SCN [1186795] instance [test11g (1)], DDL seqno
[4137].

2011-01-20 15:11:43  GGS INFO     2100  DDL operation excluded [EXCLUDE OBJNAME
myTableTemp OPTYPE CREATE], optype [CREATE], objtype [TABLE], objname
[QATEST1.MYTABLETEMP].
```

Statistics in the Process Reports

You can send current statistics for DDL processing to the Extract and Replicat reports by using the `SEND` command in GGSCI.

```
SEND {EXTRACT | REPLICAT} group REPORT
```

The statistics show totals for:

- All DDL operations
- Operations that are `MAPPED` in scope
- Operations that are `UNMAPPED` in scope
- Operations that are `OTHER` in scope
- Operations that were excluded (number of operations minus included ones)
- Errors (Replicat only)
- Retried errors (Replicat only)
- Discarded errors (Replicat only)
- Ignored operations (Replicat only)

Tracing DDL Processing

If you open a support case with Oracle GoldenGate Technical Support, you might be asked to turn on tracing. `TRACE` and `TRACE2` control DDL tracing.

Using Edition-Based Redefinition

Oracle GoldenGate supports the use of Edition-based Redefinition (EBR) with Oracle Databases enabling you to upgrade the database component of an application while it is in use, thereby minimizing or eliminating down time.

Editions are non-schema objects that Editioned objects belong to. Editions can be thought of as owning editioned objects or as a namespace. Every database starts with one edition named, `ORA$BASE`; this includes upgraded databases. More than one edition can exist in a database and each can only have one child. For example, if you create three editions in succession, edition1, edition2, edition3, then edition1 is the parent of edition2 which is the parent of edition3. This is irrespective of the user or database session that creates them or which edition was current when the new one is created. When you create an edition, it inherits all the editioned objects of its parent. To use editions with Oracle GoldenGate, you must create them.

An object is considered editioned if it is an editionable type, it is created with the `EDITIONABLE` attribute, and the schema is enabled for editioning of that object type. When you create, alter, or drop an editioned object, the redo log will contain the name of the edition in which it belongs. In a container database, editions belong to the container and each container has its own default edition.

The `CREATE | DROP EDITION` DDLs are captured for all Extract configurations. They fall into the `OTHER` category and assigned an `OBJTYPE` option value of `EDITION`. The `OBJTYPE` option can be used for filtering, for example:

```
DDL EXCLUDE OBJTYPE EDITION
DDL EXCLUDE OBJTYPE EDITION OPTYPE CREATE
DDL EXCLUDE OBJTYPE EDITION OPTYPE DROP
DDL EXCLUDE OBJTYPE EDITION OPTYPE DROP ALLOWEMPTYOWNER OBJNAME
edition_name
```

You must use the following syntax to exclude an edition from Extract or Replicat:

```
EXCLUDE OBJTYPE EDITION, ALLOWEMPTYOWNER OBJNAME edition_name
```

Editions fall into the `OTHER` category so no mapping is performed on the edition name. When applied, the `USE` permission is automatically granted to the Replicat user. Replicat will also perform a `grant use on edition` *`name`* `with grant option` to the original creating user if that user exists on the target database. Because editions are not mappable operations, they do not have owners so the standard `EXCLUDE` statement does not work.

The DDLs used to create or alter editions does not actually enable the user for editions, rather they enable the schema for editions. This is an important distinction because it means that the Replicat user does not need to be enabled for editions to apply DDLs to editioned objects. When Replicat applies a `CREATE EDITION` DDL, it grants the original creating user permission to `USE` it if the original user exists on the target database. For any unreplicated `CREATE EDITION` statements, you must issue a `USE WITH GRANT OPTION` grant to the Replicat user.

Whether or not an editionable objects becomes editioned is controlled by the schema it is applied in. Replicat switches its current session Edition before applying a DDL if the edition name attribute exists in the trail file and it is not empty.

Container database environments are supported for both Extract and Replicat. No additional configuration is necessary. The Replicat user's schema can not be enabled for editions if it is a common user. The Replicat user's schema does not need to be enabled for editions when applying DDLs to editioned objects in other schemas.

6

Instantiating Oracle GoldenGate Replication

This chapter contains instructions for configuring and performing an instantiation of the replication environment to establish and maintain a synchronized state between two or more databases. In a synchronized state, the source and target objects contain identical or appropriately corresponding values, depending on whether any conversion or transformation is performed on the data before applying it to the target objects.
Topics:

Overview of the Instantiation Process

In the instantiation procedure, you make a copy of the source data and load the copy to the target database.

The initial load captures a point-in-time snapshot of the data, while Oracle GoldenGate maintains that consistency by applying transactional changes that occur while the static data is being loaded. After instantiation is complete, Oracle GoldenGate maintains the synchronized state throughout ongoing transactional changes.

When you instantiate Oracle GoldenGate processing, it is recommended that you do so first in a test environment before deploying live on your production machines. This is especially important in an active-active or high availability configuration, where trusted source data may be touched by the replication processes. Testing enables you to find and resolve any configuration mistakes or data issues without the need to interrupt user activity for re-loads on the target or other troubleshooting activities. Testing also ensures that your instantiation process is configured properly. Parameter files can be copied to the production equipment after successful testing, and then you can perform a predictable instantiation with production data.

Prerequisites for Instantiation

The following steps must be taken before starting any Oracle GoldenGate processes or native database load processes.

- Configuring and Adding Change Synchronization Groups
- Disabling DDL Processing
- Adding Collision Handling
- Preparing the Target Tables

Configuring and Adding Change Synchronization Groups

To perform an instantiation of the target database and the replication environment, the online change capture and apply groups must exist and be properly configured. See:

- #unique_148
- Configuring Oracle GoldenGate Replicat
- #unique_149

Disabling DDL Processing

You must disable DDL activities before performing an instantiation. You can resume DDL after the instantiation is finished. See #unique_150 for instructions.

Adding Collision Handling

This prerequisite applies to the following instantiation methods:

- Configuring a Direct Bulk Load to SQL*Loader
- Configuring a Load from an Input File to SQL*Loader

This prerequisite *does not* apply to the instantiation method described in Configuring a Load with an Oracle Data Pump.

If the source database will remain active during one of those initial load methods, collision-handling logic must be added to the Replicat parameter file. This logic handles conflicts that occur because static data is being loaded to the target tables while Oracle GoldenGate replicates transactional changes to those tables.

To handle collisions, add the `HANDLECOLLISIONS` parameter to the Replicat parameter file to resolve these collisions:

- `INSERT` operations for which the row already exists
- `UPDATE` and `DELETE` operations for which the row does not exist

`HANDLECOLLISIONS` should be removed from the Replicat parameter file at the end of the instantiation steps (as prompted in the instructions).

To use the `HANDLECOLLISIONS` function to reconcile incremental data changes with the load, each target table must have a primary or unique key. If you cannot create a key through your application, use the `KEYCOLS` option of the `TABLE` and `MAP` parameters to specify columns as a substitute key for Oracle GoldenGate to use. If you cannot create keys, the affected source table must be quiesced for the load.

Preparing the Target Tables

The following are suggestions that can make the load go faster and help you to avoid errors.

- **Data**: Make certain that the target tables are empty. Otherwise, there may be duplicate-row errors or conflicts between existing rows and rows that are being loaded.
- **Indexes**: Remove indexes from the target tables. Indexes are not necessary for the inserts performed by the initial load process and will slow it down. You can add back the indexes after the load is finished.

Configuring the Initial Load for Microservices Architecture

Configuring initial load Extract can be performed from the Oracle GoldenGate MA web interface or Admin Client.

To configure from the Oracle GoldenGate MA web interface, see Add an Initial Load Extract.

To configure using the command line, see Instantiating Oracle GoldenGate with an Initial Load Using Admin Client

Performing the Target Instantiation

This procedure instantiates the target tables while Oracle GoldenGate captures ongoing transactional changes on the source and stores them until they can be applied on the target.

By the time you perform the instantiation of the target tables, the entire Oracle GoldenGate environment should be configured for change capture and delivery, as should the initial-load processes if using Oracle GoldenGate as an initial-load utility.

Note:

The first time that Extract starts in a new Oracle GoldenGate configuration, any open source transactions will be skipped. Only transactions that begin after Extract starts are captured.

- Performing Instantiation with Oracle Data Pump
- Performing Instantiation with Direct Bulk Load to SQL*Loader
- Performing Instantiation From an Input File to SQL*Loader
- Monitoring and Controlling Processing After the Instantiation
  After the target is instantiated and replication is in effect, you can control processes and view the overall health of the replication environment.

Performing Instantiation with Oracle Data Pump

To perform instantiation with Oracle Data Pump, see My Oracle Support document 1276058.1. To obtain this document, do the following:

1. Go to `http://support.oracle.com`.
2. Under Sign In, select your language and then log in with your Oracle Single Sign-On (SSO).
3. On the Dashboard, expand the Knowledge Base heading.
4. Under Enter Search Terms, paste or type the document ID of `1276058.1` and then click **Search**.
5. In the search results, select **Oracle GoldenGate Best Practices: Instantiation from an Oracle Source Database [Article ID 1276058.1]**.
6. Click the link under Attachments to open the article.

Performing Instantiation with Direct Bulk Load to SQL*Loader

1. On the source system, run GGSCI.
2. Start the primary change-capture Extract group.

   ```
   START EXTRACT group
   ```

3. Start the data-pump Extract group.

   ```
   START EXTRACT data_pump
   ```

4. If replicating sequence values:
   - Issue the `DBLOGIN` command with the alias of a user in the credential store who has `EXECUTE` privilege on `update.Sequence`.

     ```
     DBLOGIN USERIDALIAS alias
     ```

   - Issue the following command to update each source sequence and generate redo. From the redo, Replicat performs initial synchronization of the sequences on the target.

     ```
     FLUSH SEQUENCE [container.]schema.sequence
     ```

5. Start the initial-load Extract.

   ```
   START EXTRACT initial-load_Extract
   ```

   > **WARNING:**
   >
   > Do not start the initial-load Replicat. The Manager process starts it automatically and terminates it when the load is finished.

6. On the target system, run GGSCI.
7. Issue the `VIEW REPORT` command to determine when the initial load to SQL*Loader is finished.

   ```
   VIEW REPORT initial-load_Extract
   ```

8. When the load is finished, start the change-data Replicat group.

   ```
   START REPLICAT group
   ```

9. Issue the `INFO REPLICAT` command, and continue to issue it until it shows that Replicat posted all of the change data that was generated during the initial load. For example, if the initial-load Extract stopped at 12:05, make sure Replicat posted data up to that time.

   ```
   INFO REPLICAT group
   ```

10. Turn off `HANDLECOLLISIONS` for the change-delivery Replicat to disable initial-load error handling.

    ```
    SEND REPLICAT group, NOHANDLECOLLISIONS
    ```

11. Edit the change-delivery Replicat parameter file to remove the `HANDLECOLLISIONS` parameter.

    ```
    EDIT PARAMS group
    ```

12. Save and close the parameter file.

From this point forward, Oracle GoldenGate continues to synchronize data changes.

Performing Instantiation From an Input File to SQL*Loader

Note:

The SQL*Loader method is not recommended if the data has multibyte characters, especially when the character set of the operating system is different from the database character set.

1. On the source system, run GGSCI.
2. Start the primary change-capture Extract group.

   ```
   START EXTRACT group
   ```

3. Start the data-pump Extract group.

   ```
   START EXTRACT data_pump
   ```

4. If replicating sequence values:
   - Issue the `DBLOGIN` command with the alias of a user in the credential store who has `EXECUTE` privilege on `update.Sequence`.

     ```
     DBLOGIN USERIDALIAS alias
     ```

   - Issue the following command to update each source sequence and generate redo. From the redo, Replicat performs initial synchronization of the sequences on the target.

     ```
     FLUSH SEQUENCE [container.]schema.sequence
     ```

5. From the Oracle GoldenGate installation directory on the source system, start the initial-load Extract from the command line of the operating system (not GGSCI).

   UNIX and Linux:

```
$ /OGG_directory/extract paramfile dirprm/initial-load_Extract.prm
reportfile path
```

Windows:

```
C:\> OGG_directory\extract paramfile dirprm\initial-load_Extract.prm
reportfile path
```

Where: *`initial-load_Extract`* is the name of the initial-load Extract and *`path`* is the relative or fully qualified path where you want the Extract report file to be created.

6. Wait until the initial extraction from the source is finished. Verify its progress and results by viewing the Extract report file from the command line.
7. On the target system, start the initial-load Replicat.

   UNIX and Linux:

   ```
   $ /OGG directory/replicat paramfile dirprm/initial-load_Replicat
   name.prm reportfile path
   ```

   Windows:

   ```
   C:\> OGG directory\replicat paramfile dirprm\initial-load_Replicat.prm
   reportfile path
   ```

   Where: `initial-load Extract` is the name of the initial-load Replicat and *`path`* is the relative or fully qualified path where you want the Replicat report file to be created.

8. When the initial-load Replicat stops, verify its results by viewing the Replicat report file from the command line.
9. Using the ASCII-formatted file and the run and control files that the initial-load Replicat created, load the data with SQL*Loader.
10. When the load is finished, start the change-delivery Replicat group.

    ```
    START REPLICAT group
    ```

11. Issue the `INFO REPLICAT` command, and continue to issue it until it shows that Replicat posted all of the change data that was generated during the initial load. For example, if the initial-load Extract stopped at 12:05, make sure Replicat posted data up to that time.

    ```
    INFO REPLICAT group
    ```

12. Turn off `HANDLECOLLISIONS` for the change-delivery Replicat to disable initial-load error handling.

    ```
    SEND REPLICAT group, NOHANDLECOLLISIONS
    ```

13. Edit the change-delivery Replicat parameter file to remove the `HANDLECOLLISIONS` parameter.

    ```
    EDIT PARAMS group
    ```

14. Save and close the parameter file.

From this point forward, Oracle GoldenGate continues to synchronize data changes.

Monitoring and Controlling Processing After the Instantiation

After the target is instantiated and replication is in effect, you can control processes and view the overall health of the replication environment.

If you configured Replicat in integrated mode, you can use the `STATS REPLICAT` command to view statistics on the number of transactions that are applied in integrated mode as compared to those that are applied in direct apply mode.

```
STATS REPLICAT group
```

The output of this command shows the number of transactions applied, the number of transactions that were redirected to direct apply, and the direct transaction ratio, among other statistics. The statistics help you determine whether integrated Replicat is performing as intended. If the environment is satisfactory and there is a high ratio of direct apply operations, consider using nonintegrated Replicat. You can configure parallelism with nonintegrated Replicat.

Note:

To ensure realistic statistics, view apply statistics only after you are certain that the Oracle GoldenGate environment is well established, that configuration errors are resolved, and that any anticipated processing errors are being handled properly.

You can also view runtime statistics for integrated Replicat in the `V$`views for each of the inbound server components.

- The reader statistics are recorded in `V$GG_APPLY_READER` and include statistics on number of messages read, memory used, and dependency counts.
- The apply coordinator statistics are recorded in `V$GG_APPLY_COORDINATOR` and record statistics at the transaction level.
- The apply server statistics are recorded in `V$GG_APPLY_SERVER`. This view records information for each of the apply server processes (controlled by `parallelism` and `max_parallelism` parameters) as separate rows. The statistics for each apply server are identified by the `SERVER_ID` column. If a `SERVER_ID` of `0` exists, this represents an aggregate of any apply servers that exited because the workload was reduced.
- Statistics about the number of messages received by the database from Replicat are recorded in the `V$GG_APPLY_RECEIVER` table.

To control processes, see Controlling Oracle GoldenGate Processes in *Administering Oracle GoldenGate*.

To ensure that all processes are running properly and that errors are being handled according to your error handling rules, see Handling Processing Errors in *Administering Oracle GoldenGate*. Oracle GoldenGate provides commands and logs to view process status, lag, warnings, and other information.

To know more about querying the following views, see Oracle Database Reference.

- `V$GOLDENGATE_TABLE_STATS` to see statistics for DML and collisions that occurred for each replicated table that the inbound server processed.

- `V$GOLDENGATE_TRANSACTION` to see information about transactions that are being processed by Oracle GoldenGate inbound servers.

Verifying Synchronization

To verify that the source and target data are synchronized, you can use the Oracle GoldenGate Veridata product or use your own scripts to select and compare source and target data.

Backing up the Oracle GoldenGate Environment

After you start Oracle GoldenGate processing, an effective backup routine is critical to preserving the state of processing in the event of a failure. Unless the Oracle GoldenGate working files can be restored, the entire replication environment must be re-instantiated, complete with new initial loads.

As a best practice, include the entire Oracle GoldenGate home installation in your backup routines. There are too many critical sub-directories, as well as files and programs at the root of the directory, to keep track of separately. In any event, the most critical files are those that consume the vast majority of backup space, and therefore it makes sense just to back up the entire installation directory for fast, simple recovery.

7

Configuring a Downstream Mining Database

This appendix contains instructions for preparing a downstream Oracle mining database to support Extract.
For examples of the downstream mining configuration, see #unique_158.

Topics:

- Evaluating Capture Options for a Downstream Deployment
  Downstream deployment allows you to offload the source database.
- Preparing the Source Database for Downstream Deployment
  The source database ships its redo logs to a downstream database, and Extract uses the logmining server at the downstream database to mine the redo logs.
- Preparing the Downstream Mining Database
  A downstream mining database can accept both archived logs and online redo logs from a source database.
- Enabling Downstream Extract Registration Using ADG Redirection in Downstream Configuration
  Oracle GoldenGate supports downstream Extract registration using ADG redirection in a downstream mining database configuration.
- Example 1: Capturing from One Source Database in Real-time Mode
  This example captures changes from source database DBMS1 by deploying an Extract at a downstream mining database DBMSCAP.
- Example 2: Capturing from Multiple Sources in Archive-log-only Mode
  The following example captures changes from database DBMS1 and DBMS2 by deploying an Extract at a downstream mining database DBMSCAP.
- Example 3: Capturing from Multiple Sources with Mixed Real-time and Archive-log-only Mode
  The following example captures changes from database DBMS1, DBMS2 and DBMS3 by deploying an Extract at a downstream mining database DBMSCAP.

Evaluating Capture Options for a Downstream Deployment

Downstream deployment allows you to offload the source database.

A downstream mining database can accept both archived logs and online redo logs from a source database.

Multiple source databases can send their redo data to a single downstream database; however the downstream mining database can accept *online* redo logs from only one of those source databases. The rest of the source databases must ship archived logs.

When online logs are shipped to the downstream database, *real-time capture* by Extract is possible. Changes are captured as though Extract is reading from the source logs. In order to accept online redo logs from a source database, the downstream mining database must have standby redo logs configured.

When using a downstream mining configuration, the source database and mining database must be the same endian and same bitsize, which is 64 bits. For example, if the source database was on Linux 64-bit, you can have the mining database run on Windows 64-bit, because they have the same endian and bitsize.

Preparing the Source Database for Downstream Deployment

The source database ships its redo logs to a downstream database, and Extract uses the logmining server at the downstream database to mine the redo logs.

This section guides you in the process of:

- Creating the Source User Account
  There must be an Extract user on the source database. Extract uses the credentials of this user to do metadata queries and to fetch column values as needed from the source database.
- Configuring Redo Transport from Source to Downstream Mining Database
  To set up the transfer of redo log files from a source database to the downstream mining database, and to prepare the downstream mining database to accept these redo log files, perform the steps given in this topic.

Creating the Source User Account

There must be an Extract user on the source database. Extract uses the credentials of this user to do metadata queries and to fetch column values as needed from the source database.

The source user is specified by the `USERIDALIAS` parameter.

To assign the required privileges, follow the procedure in Establishing Oracle GoldenGate Credentials

Configuring Redo Transport from Source to Downstream Mining Database

To set up the transfer of redo log files from a source database to the downstream mining database, and to prepare the downstream mining database to accept these redo log files, perform the steps given in this topic.

The following summarizes the rules for supporting multiple sources sending redo to a single downstream mining database:

- Only one source database can be configured to send *online* redo to the standby redo logs at the downstream mining database. The `log_archive_dest_n` setting for this source database should *not* have a `TEMPLATE` clause.
- Source databases that are *not* sending online redo to the standby redo logs of the downstream mining database *must have* a `TEMPLATE` clause specified in the `log_archive_dest_n` parameter.
- Each of the source databases that sends redo to the downstream mining database must have a unique `DBID`. You can select the `DBID` column from the `v$database` view of these source databases to ensure that the DBIDs are unique.

- The `FAL_SERVER` value must be set to the downstream mining database. `FAL_SERVER` specifies the FAL (fetch archive log) server for a standby database. The value is a list of Oracle Net service names, which are assumed to be configured properly on the standby database system to point to the desired FAL servers. The list contains the net service name of any database that can potentially ship redo to the downstream database.
- When using redo transport, there could be a delay in processing redo due to network latency. For Extract, this latency is monitored by measuring the delay between LCRs received from source database and reporting it. If the latency exceeds a threshold, a warning message appears in the report file and a subsequent information message appears when the lag drops to normal values. The default value for the threshold is 10 seconds.

Note:

The archived logs shipped from the source databases are called *foreign archived logs*. You must not use the recovery area at the downstream mining database to store foreign archived logs. Such a configuration is not supported by Extract. Foreign archived logs stored in the Flash Recovery Area (FRA) are not automatically deleted by RMAN jobs. These archived logs must be manually purged.

These instructions take into account the requirements to ship redo from multiple sources, if required. You must configure an Extract process for each of those sources.

To Configure Redo Transport

1. Configure Oracle Net so that each source database can communicate with the mining database. For instructions, see *Oracle Database Net Services Administrator's Guide*.
2. Configure authentication at each source database and at the downstream mining database to support the transfer of redo data. Redo transport sessions are authenticated using either the Secure Sockets Layer (SSL) protocol or a remote login password file. If a source database has a remote login password file, copy it to the appropriate directory of the mining database system. The password file must be the same at all source databases, and at the mining database. For more information about authentication requirements for redo transport, see Preparing the Primary Database for Standby Database Creation in *Oracle Data Guard Concepts and Administration*.
3. At each source database, configure one `LOG_ARCHIVE_DEST_n` initialization parameter to transmit redo data to the downstream mining database. Set the attributes of this parameter as shown in one of the following examples, depending on whether real-time or archived-log-only capture mode is to be used.
    - Example for real-time capture at the downstream logmining server, where the source database sends its online redo logs to the downstream database:

      ```
      ALTER SYSTEM
      SET LOG_ARCHIVE_DEST_2='SERVICE=DBMSCAP.EXAMPLE.COM ASYNC NOREGISTER
      VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)DB_UNIQUE_NAME=dbmscap'
      ```
    - Example for archived-log-only capture at the downstream logmining server:

      ```
      ALTER SYSTEM SET
      LOG_ARCHIVE_DEST_2='SERVICE=DMBSCAP.EXAMPLE.COM ASYNC NOREGISTER
      VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)
      ```

```
TEMPLATE=/usr/oracle/log_for_dbms1/dbms1_arch_%t_%s_%r.log
DB_UNIQUE_NAME=dbmscap'
```

Note:

When using an archived-log-only downstream mining database, you must specify a value for the `TEMPLATE` attribute. Oracle also recommends that you use the `TEMPLATE` clause in the source databases so that the log files from all remote source databases are kept separated from the local database log files, and from each other.

4. At the source database, set a value of `ENABLE` for the `LOG_ARCHIVE_DEST_STATE_n` initialization parameter that corresponds with the `LOG_ARCHIVE_DEST_n` parameter that corresponds to the destination for the downstream mining database, as shown in the following example.

   ```
   ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2=ENABLE
   ```

5. At the source database, and at the downstream mining database, set the `DG_CONFIG` attribute of the `LOG_ARCHIVE_CONFIG` initialization parameter to include the `DB_UNIQUE_NAME` of the source database and the downstream database, as shown in the following example.

   ```
   ALTER SYSTEM SET LOG_ARCHIVE_CONFIG='DG_CONFIG=(dbms1,dbmscap)'
   ```

Preparing the Downstream Mining Database

A downstream mining database can accept both archived logs and online redo logs from a source database.

The following sections explain how to prepare the downstream mining database:

- Creating the Downstream Mining User Account
- Registering Extract with the Mining Database
  You need to create a database logmining server to capture redo data when using a downstream database.
- Configuring the Mining Database to Archive Local Redo Log Files
- Configure the Wallet for the Downstream Mining Database
- Preparing a Downstream Mining Database for Real-time Capture

Creating the Downstream Mining User Account

When using a downstream mining configuration, there must be an Extract mining user on the downstream database. The mining Extract process uses the credentials of this user to interact with the downstream logmining server. The downstream mining user is specified by the `TRANLOGOPTIONS` parameter with the `MININGUSERALIAS` option. See Establishing Oracle GoldenGate Credentials to assign the correct credentials for the version of your database.

Registering Extract with the Mining Database

You need to create a database logmining server to capture redo data when using a downstream database.

The creation of the logmining server captures a snapshot of the source database in the redo stream of the source database. In a source multitenant container database, you register Extract with each of the pluggable databases that you want to include for Extract.

> **WARNING:**
>
> Make certain that you know the earliest SCN of the log stream at which you want Extract to begin processing. Extract cannot have a starting SCN value that is lower than the first SCN that is specified when the underlying database capture process is created with the `REGISTER EXTRACT` command. You can use the `SCN` option

1. Log into the mining database then use the commands appropriate to your environment. The use of `DBLOGIN` always refers to the source database.

   Command for source database deployment:

   ```
   DBLOGIN USERIDALIAS ggeast
   ```

   Command for downstream mining database deployment:

   ```
   DBLOGIN USERIDALIAS ggwest
   MININGDBLOGIN USERIDALIAS dbnorth
   ```

   Where: *`alias`* specifies the alias of the database login credential that is assigned to Extract. This credential must exist in the Oracle GoldenGate credential store. For more information, see Establishing Oracle GoldenGate Credentials. For more information about `DBLOGIN` and `MININGDBLOGIN`, see *Reference for Oracle GoldenGate*.

2. Register the Extract process with the mining database.

   ```
   REGISTER EXTRACT group DATABASE [CONTAINER (container[, ...])] [SCN
   system_change_number]
   ```

   Where:

   - *`group`* is the name of the Extract group.
   - `CONTAINER (container[, ...])` specifies a pluggable database (PDB) within a multitenant container database, or a list of PDBs separated with commas. The specified PDBs must exist before the `REGISTER` command is executed. Extract will capture only from the PDBs that are listed in this command. For example, the following command registers PDBs `mypdb1` and `mypdb4`. Changes from any other PDBs in the multitenant container database are ignored by Oracle GoldenGate.

     ```
     REGISTER EXTRACT exte DATABASE CONTAINER (pdbeast, pdbwest, dbnorth)
     ```

     You can add or drop pluggable databases at a later date by stopping Extract, issuing a `DBLOGIN` command, and then issuing `REGISTER EXTRACT` with the `{ADD | DROP} CONTAINER` option of `DATABASE`.

> **Note:**
>
> Adding CONTAINERs at particular SCN on an existing Extract is not supported.

- Registers Extract to begin capture at a specific SCN in the past. Without this option, capture begins from the time that REGISTER EXTRACT is issued. The specified SCN must correspond to the begin SCN of a dictionary build operation in a log file. You can issue the following query to find all valid SCN values:

```
SELECT first_change#
   FROM v$archived_log
   WHERE dictionary_begin = 'YES' AND
      STANDBY_DEST = 'NO' AND
      NAME IS NOT NULL AND
      STATUS = 'A';
```

3. To register additional Extracts with a downstream database for the same source database, issue this REGISTER command.

   If you want to have more than one extract per source database, you can do that using the SHARE with REGISTER EXTRACT for better performance and metadata management. The specified SCN must correspond to the SCN where mining should begin in the archive logs.

```
REGISTER EXTRACT group DATABASE [CONTAINER (container[, ...])]
[SCN system_change_number] SHARE
```

> **Note:**
>
> The register process may take a few to several minutes to complete, even though the REGISTER command returns immediately.

Configuring the Mining Database to Archive Local Redo Log Files

This procedure configures the downstream mining database to archive redo data in its online redo logs. These are redo logs that are generated at the downstream mining database.

Archiving must be enabled at the downstream mining database if you want to run Extract in real-time integrated capture mode, but it is also recommended for archive-log-only capture. Extract in integrated capture mode writes state information in the database. Archiving and regular backups will enable you to recover this state information in case there are disk failures or corruption at the downstream mining database.

To Archive Local Redo Log Files

1. Alter the downstream mining database to be in archive log mode. You can do this by issuing the following DDL.

   ```
   STARTUP MOUNT;
   ALTER DATABASE ARCHIVELOG;
   ALTER DATABASE OPEN;
   ```

2. At the downstream mining database, set the first archive log destination in the `LOG_ARCHIVE_DEST_n` initialization parameter as shown in the following example:

   ```
   ALTER SYSTEM SET
   LOG_ARCHIVE_DEST_1='LOCATION=/home/arc_dest/local
   VALID_FOR=(ONLINE_LOGFILE,PRIMARY_ROLE)'
   ```

   Alternatively, you can use a command like this example:

   ```
   ALTER SYSTEM SET
   LOG_ARCHIVE_DEST_1='LOCATION='USE_DB_RECOVERY_FILE_DEST'
   valid_for=(ONLINE_LOGFILE,PRIMARY_ROLE)'
   ```

   > **Note:**
   >
   > The online redo logs generated by the downstream mining database can be archived to a recovery area. However, you must not use the recovery area of the downstream mining database to stage foreign archived logs or to archive standby redo logs. For information about configuring a fast recovery area, see the *Oracle Database Backup and Recovery User's Guide*.

3. Enable the local archive destination.

   ```
   ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_1=ENABLE
   ```

For more information about these initialization parameters, see Set Primary Database Initialization Parameters in the *Oracle Data Guard Concepts and Administration* guide.

Configure the Wallet for the Downstream Mining Database

When TDE is enabled on source database and downstream database, then the source wallet or keys should be the same on the downstream mining database and the source database.

Follow these steps to copy the wallet directory from the source database to the downstream mining database:

1. Shutdown the downstream database using the `shutdown immediate` command.
2. Remove the wallet directory on downstream database view:

   ```
   rm $T_WORK/wallet/*
   ```

3. Copy the `$T_WORK/wallet/*` from the source database view to the downstream database view.
4. Restart the downstream database.
5. Run checksum on the source database view and downstream database view to ensure that it matches:

   ```
   cksum $T_WORK/wallet/*
   ```

Preparing a Downstream Mining Database for Real-time Capture

This procedure is only required if you want to use real-time capture at a downstream mining database. It is not required to use archived-log-only capture mode. To use real-time capture, it is assumed that the downstream database has already been configured to archive its local redo data as shown in Configuring the Mining Database to Archive Local Redo Log Files .

- Create the Standby Redo Log Files
- Configure the Database to Archive Standby Redo Log Files Locally

Create the Standby Redo Log Files

The following steps outline the procedure for adding standby redo log files to the downstream mining database. The following summarizes the rules for creating the standby redo logs:

- Each standby redo log file must be at least as large as the largest redo log file of the redo source database. For administrative ease, Oracle recommends that all redo log files at source database and the standby redo log files at the downstream mining database be of the same size.
- The standby redo log must have at least one more redo log group than the redo log at the source database, for each redo thread at the source database.

The specific steps and SQL statements that are required to add standby redo log files depend on your environment. See Oracle Data Guard Concepts and Administration 11g Release 2 (11.2) for detailed instructions about adding standby redo log files to a database.

> **Note:**
>
> If there will be multiple source databases sending redo to a single downstream mining database, only one of those sources can send redo to the standby redo logs of the mining database. An Extract process that mines the redo from this source database can run in real-time mode. All other source databases must send only their archived logs to the downstream mining database, and the Extracts that read this data must be configured to run in archived-log-only mode.

To Create the Standby Redo Log Files

1. In SQL*Plus, connect to the source database as an administrative user.

2. Determine the size of the source log file. Make note of the results.

   ```
   SELECT BYTES FROM V$LOG;
   ```

3. Determine the number of online log file groups that are configured on the source database. Make note of the results.

   ```
   SELECT COUNT(GROUP#) FROM V$LOG;
   ```

4. Connect to the downstream mining database as an administrative user.

5. Add the standby log file groups to the mining database. The standby log file size must be at least the size of the source log file size. The number of standby log file groups must be at least one more than the number of source online log file groups. This applies to each instance (thread) in a RAC installation. So if you have "n" threads at the source database, each having "m" redo log groups, you should configure n*(m+1) redo log groups at the downstream mining database.

   The following example shows three standby log groups.

   ```
   ALTER DATABASE ADD STANDBY LOGFILE GROUP 3
   ('/oracle/dbs/slog3a.rdo', '/oracle/dbs/slog3b.rdo') SIZE 500M;
   ALTER DATABASE ADD STANDBY LOGFILE GROUP 4
   ('/oracle/dbs/slog4.rdo', '/oracle/dbs/slog4b.rdo') SIZE 500M;
   ALTER DATABASE ADD STANDBY LOGFILE GROUP 5
   ('/oracle/dbs/slog5.rdo', '/oracle/dbs/slog5b.rdo') SIZE 500M;
   ```

6. Confirm that the standby log file groups were added successfully.

   ```
   SELECT GROUP#, THREAD#, SEQUENCE#, ARCHIVED, STATUS
   FROM V$STANDBY_LOG;
   ```

   The output should be similar to the following:

   ```
   GROUP#     THREAD#    SEQUENCE#  ARC STATUS
   ---------- ---------- ---------- --- ----------
            3          0          0 YES UNASSIGNED
            4          0          0 YES UNASSIGNED
            5          0          0 YES UNASSIGNED
   ```

7. Ensure that log files from the source database are appearing in the location that is specified in the `LOCATION` attribute of the local `LOG_ARCHIVE_DEST_n` that you set. You might need to switch the log file at the source database to see files in the directory.

Configure the Database to Archive Standby Redo Log Files Locally

This procedure configures the downstream mining database to archive the standby redo logs that receive redo data from the online redo logs of the source database. Keep in mind that foreign archived logs should not be archived in the recovery area of the downstream mining database.

To Archive Standby Redo Logs Locally

1. At the downstream mining database, set the second archive log destination in the `LOG_ARCHIVE_DEST_n` initialization parameter as shown in the following example.

   ```
   ALTER SYSTEM SET
   LOG_ARCHIVE_DEST_2='LOCATION=/home/arc_dest/srl_dbms1
   VALID_FOR=(STANDBY_LOGFILE,PRIMARY_ROLE)'
   ```

   Oracle recommends that foreign archived logs (logs from remote source databases) be kept separate from local mining database log files, and from each other. You must not use the recovery area of the downstream mining database to stage foreign archived logs..

2. Enable the `LOG_ARCHIVE_DEST_2` parameter you set in the previous step as shown in the following example.

```
ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2=ENABLE
```

Enabling Downstream Extract Registration Using ADG Redirection in Downstream Configuration

Oracle GoldenGate supports downstream Extract registration using ADG redirection in a downstream mining database configuration.

This approach uses an Active Dataguard (ADG) configured in a cascaded mode to transport redo logs to a downstream mining database to use with downstream Extract, which reduces the overhead on the source database.

Extract must be started using sourceless option so that it does not connect to source database instead connects to ADG using `FETCHUSERID` or `FETCHUSERIDALIAS` when it needs to fetch any non-native datatypes.

During register, Oracle GoldenGate connects to ADG as source database instead of the database where redo originates. ADG redirection is supported for the following commands and parameters in Admin Client and GGSCI:

Note:

`SCHEMATRANDATA` and `TRANDATA`, even though the command is executed on the Standby, the actual log groups are created and maintained on the primary database where the actual DML operations take place.

- `SCHEMATRANDATA`
- `TRANDATA`
- `FLUSH SEQUENCE`
- `TRACETABLE`
- `HEARTBEATTABLE`
- `REGISTER EXTRACT`

This feature is supported for CDB and supports wildcard registration. It is only supported when using Oracle Database 21c and higher.

- How to Enable Downstream Extract Registration Using ADG Redirection

How to Enable Downstream Extract Registration Using ADG Redirection

Here are the steps to enable downstream Extract to work with ADG Standby:

1. Add an additional `LOG_ARCHIVE_DESTINATION_N (LAD)` on the ADG standby, as shown in the following example:

   ```
   alter system set log_archive_dest_3='service=service name mining db ASYNC
   NOREGISTER VALID_FOR(STANDBY_LOGFILES,STANDBY_ROLES) DB_UNIQUE_NAME=db
   unique name of 3rd db' scope=both
   ```

   This step transports and generates the `standby_logfiles` for an ADG Standby.
2. Set the `LOG_ARCHIVE_CONFIG` on the ADG Standby to ship the logs to the mining database, as shown in the following example:

   ```
   ALTER SYSTEM SET LOG_ARCHIVE_CONFIG='dg_config(db unique name of 1st
   db,db unique name of 2nd db,db unique name of 3rd db)' scope=both;
   ```

3. On the mining database, set up the location to store the incoming `standby_logfiles` on the mining database:

   ```
   alter system set log_archive_dest_2='location=USE_DB_RECOVERY_FILE_DEST
   VALID_FOR=(STANDBY_LOGFILE,ALL_ROLES)' scope=both
   ```

4. Run `LOG_ARCHIVE_CONFIG` on the mining database, so that the Extract process is able to read them on the mining database, as shown in the following example:

   ```
   ALTER SYSTEM SET LOG_ARCHIVE_CONFIG='dg_config(db unique name of 1st db,
   db unique name of 2nd db, db unique name of 3rd db)' scope=both
   ```

5. For a downstream Extract, you need to ensure that the database connections are appropriately configured for GGSCI and Admin Client. When registering the Extract, you need to make sure that `DBLOGIN` connection is made to the ADG Standby, that is open for read only activity. To add the Extract and register it, use the following command:

   ```
   dblogin userid ggadmin@inst2, password ggadmin (inst2 is the ADG not
   primary)
           miningdblogin userid ggadmin@inst3, password ggadmin (inst3 is
   the mining database)
   ```

6. Now, register an Extract that uses the `NOUSERID` parameter:

   ```
   add extract ext1, integrated tranlog, begin now register extract ext1
   database
   ```

7. After the Extract is registered, you can use this Extract to mine data and start the Extract normally.

Example 1: Capturing from One Source Database in Real-time Mode

This example captures changes from source database DBMS1 by deploying an Extract at a downstream mining database DBMSCAP.

Note:

The example assumes that you created the necessary standby redo log files as shown in Configuring a Downstream Mining Database .

This assumes that the following users exist:

- User GGADM1 in DBMS1 whose credentials Extract will use to fetch data and metadata from DBMS1. This user has the alias of `ggadm1` in the Oracle GoldenGate credential store and logs in as `ggadm1@dbms1`. It is assumed that the `DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE()` procedure was called to grant appropriate privileges to this user at the source database.
- User GGADMCAP in DBMSCAP whose credentials Extract will use to retrieve logical change records from the logmining server at the downstream mining database DBMSCAP. This user has the alias of `ggadmcap` in the Oracle GoldenGate credential store and logs in as `ggadmcap@dbmscap`. It is assumed that the `DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE()` procedure was called to grant appropriate privileges to this user at the mining database.
- Prepare the Mining Database to Archive its Local Redo
- Prepare the Mining Database to Archive Redo Received in Standby Redo Logs from the Source Database
- Prepare the Source Database to Send Redo to the Mining Database
- Set up Extract (ext1) on DBMSCAP

Prepare the Mining Database to Archive its Local Redo

To prepare the mining database to archive its local redo:

1. The downstream mining database must be in archive log mode. You can do this by issuing the following DDL.

   ```
   STARTUP MOUNT;
   ALTER DATABASE ARCHIVELOG;
   ALTER DATABASE OPEN;
   ```

2. At the downstream mining database, set `log_archive_dest_1` to archive local redo.

   ```
   ALTER SYSTEM SET
   LOG_ARCHIVE_DEST_1='LOCATION=/home/arc_dest/local
   VALID_FOR=(ONLINE_LOGFILE, PRIMARY_ROLE)'
   ```

3. Enable `log_archive_dest_1`.

```
ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_1=ENABLE
```

Prepare the Mining Database to Archive Redo Received in Standby Redo Logs from the Source Database

To prepare the mining database to archive the redo received in standby redo logs from the source database:

1. At the downstream mining database, set `log_archive_dest_2` as shown in the following example.

   ```
   ALTER SYSTEM SET
   LOG_ARCHIVE_DEST_2='LOCATION=/home/arc_dest/srl_dbms1
   VALID_FOR=(STANDBY_LOGFILE,PRIMARY_ROLE)'
   ```

2. Enable `log_archive_dest_2` as shown in the following example.

   ```
   ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2=ENABLE
   ```

3. Set `DG_CONFIG` at the downstream mining database.

   ```
   ALTER SYSTEM SET LOG_ARCHIVE_CONFIG='DG_CONFIG=(dbms1,dbmscap)'
   ```

Prepare the Source Database to Send Redo to the Mining Database

To prepare the source database to send redo to the mining database:

1. Make sure that the source database is running with the required compatibility.

   ```
   select name, value from v$parameter where name = 'compatible';

   NAME            VALUE
   ---------       ---------------------
   compatible      11.1.0.7.0
   ```

   The minimum compatibility setting required from integrated capture is 11.1.0.0.0.

2. Set `DG_CONFIG` at the source database.

   ```
   ALTER SYSTEM SET LOG_ARCHIVE_CONFIG='DG_CONFIG=(dbms1,dbmscap)';
   ```

3. Set up redo transport at the source database.

   ```
   ALTER SYSTEM
   SET LOG_ARCHIVE_DEST_2='SERVICE=DBMSCAP.EXAMPLE.COM ASYNC OPTIONAL NOREGISTER
   VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)DB_UNIQUE_NAME=dbmscap';
   ```

4. Enable the downstream destination.

   ```
   ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2=ENABLE;
   ```

Set up Extract (ext1) on DBMSCAP

To set up Extract (ext1) on DBMSCAP:

1. Register Extract with the downstream mining database. In the credential store, the alias name of `ggadm1` is linked to a user connect string of `ggadm1@dbms1`. The alias name of `ggadmcap` is linked to a user connect string of `ggadmcap@dbmscap`.

```
GGSCI> DBLOGIN USERIDALIAS ggadm1
GGSCI> MININGDBLOGIN USERIDALIAS ggadmcap
GGSCI> REGISTER EXTRACT ext1 DATABASE
```

2. Create Extract at the downstream mining database.

   ```
   GGSCI> ADD EXTRACT ext1 INTEGRATED TRANLOG BEGIN NOW
   ```

3. Edit Extract parameter file `ext1.prm`. The following lines must be present to take advantage of real-time capture. In the credential store, the alias name of `ggadm1` is linked to a user connect string of `ggadm1@dbms1`. The alias name of `ggadmcap` is linked to a user connect string of `ggadmcap@dbmscap`.

   ```
   USERIDALIAS ggadm1
   TRANLOGOPTIONS MININGUSERALIAS ggadmcap
   TRANLOGOPTIONS INTEGRATEDPARAMS (downstream_real_time_mine Y)
   ```

4. Start Extract.

   ```
   GGSCI> START EXTRACT ext1
   ```

Note:

You can create multiple Extracts running in real-time Extract mode in the downstream mining database, as long as they all are capturing data from the same source database, such as capturing changes for database DBMS1 in the preceding example.

Example 2: Capturing from Multiple Sources in Archive-log-only Mode

The following example captures changes from database DBMS1 and DBMS2 by deploying an Extract at a downstream mining database DBMSCAP.

It assumes the following users:

- User GGADM1 in DBMS1 whose credentials Extract will use to fetch data and metadata from DBMS1. It is assumed that the `DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE()` procedure was called to grant appropriate privileges to this user at DBMS1.
- User GGADM2 in DBMS2 whose credentials Extract will use to fetch data and metadata from DBMS2. It is assumed that the `DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE()` procedure was called to grant appropriate privileges to this user at DBMS2.
- User GGADMCAP in DBMSCAP whose credentials Extract will use to retrieve logical change records from the logmining server at the downstream mining database. It is assumed that the `DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE()` procedure was called to grant appropriate privileges to this user at the downstream mining database DBMSCAP.

This procedure also assumes that the downstream mining database is configured in archive log mode.

- Prepare the Mining Database to Archive its Local Redo

- Prepare the Mining Database to Archive Redo from the Source Database
- Prepare the First Source Database to Send Redo to the Mining Database
- Prepare the Second Source Database to Send Redo to the Mining Database
- Set up Extracts at Downstream Mining Database

Prepare the Mining Database to Archive its Local Redo

To prepare the mining database to archive its local redo:

1. The downstream mining database must be in archive log mode. You can do this by issuing the following DDL.

   ```
   STARTUP MOUNT;
   ALTER DATABASE ARCHIVELOG;
   ALTER DATABASE OPEN;
   ```

2. At the downstream mining database, set `log_archive_dest_1` to archive local redo.

   ```
   ALTER SYSTEM SET
   LOG_ARCHIVE_DEST_1='LOCATION=/home/arc_dest/local
   VALID_FOR=(ONLINE_LOGFILE, PRIMARY_ROLE)'
   ```

3. Enable `log_archive_dest_1`.

   ```
   ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_1=ENABLE
   ```

Prepare the Mining Database to Archive Redo from the Source Database

Set `DG_CONFIG` at the downstream mining database.

```
ALTER SYSTEM SET LOG_ARCHIVE_CONFIG='DG_CONFIG=(dbms1,dbms2, dbmscap)'
```

Prepare the First Source Database to Send Redo to the Mining Database

To prepare the first source database to send redo to the mining database:

1. Make certain that DBMS1 source database is running with the required compatibility.

   ```
   select name, value from v$parameter where name = 'compatible';

   NAME            VALUE
   ---------       ---------------------
   compatible      11.1.0.0.0
   ```

   The minimum compatibility setting required from capture is 11.1.0.0.0.

2. Set `DG_CONFIG` at DBMS1 source database.

   ```
   ALTER SYSTEM SET LOG_ARCHIVE_CONFIG='DG_CONFIG=(dbms1, dbmscap)';
   ```

3. Set up redo transport at DBMS1 source database. The `TEMPLATE` clause is mandatory if you want to send redo data directly to foreign archived logs at the downstream mining database.

   ```
   ALTER SYSTEM
   SET LOG_ARCHIVE_DEST_2='SERVICE=DBMSCAP.EXAMPLE.COM ASYNC OPTIONAL NOREGISTER
   TEMPLATE='/usr/orcl/arc_dest/dbms1/dbms1_arch_%t_%s_%r.log
   VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)DB_UNIQUE_NAME=dbmscap';
   ```

4. Enable the downstream destination.

```
ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2=ENABLE;
```

Prepare the Second Source Database to Send Redo to the Mining Database

To prepare the second source database to send redo to the mining database:

1. Make sure that DBMS2 source database is running with the required compatibility.

   ```
   select name, value from v$parameter where name = 'compatible';

   NAME                    VALUE
   ---------               ---------------------
   compatible              11.1.0.0.0
   ```

   The minimum compatibility setting required from capture is 11.1.0.0.0.

2. Set `DG_CONFIG` at DBMS2 source database.

   ```
   ALTER SYSTEM SET LOG_ARCHIVE_CONFIG='DG_CONFIG=(dbms2, dbmscap)';
   ```

3. Set up redo transport at DBMS2 source database. The `TEMPLATE` clause is mandatory if you want to send redo data directly to foreign archived logs at the downstream mining database.

   ```
   ALTER SYSTEM
   SET LOG_ARCHIVE_DEST_2='SERVICE=DBMSCAP.EXAMPLE.COM ASYNC OPTIONAL NOREGISTER
   TEMPLATE='/usr/orcl/arc_dest/dbms2/dbms2_arch_%t_%s_%r.log
   VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)DB_UNIQUE_NAME=dbmscap';
   ```

4. Enable the downstream destination.

   ```
   ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2=ENABLE;
   ```

Set up Extracts at Downstream Mining Database

These steps set up Extract at the downstream database to capture from the archived logs sent by DBMS1 and DBMS2.

Example 3: Capturing from Multiple Sources with Mixed Real-time and Archive-log-only Mode

The following example captures changes from database DBMS1, DBMS2 and DBMS3 by deploying an Extract at a downstream mining database DBMSCAP.

Note:

This example assumes that you created the necessary standby redo log files as shown in Configuring a Downstream Mining Database .

It assumes the following users:

- User GGADM1 in DBMS1 whose credentials Extract will use to fetch data and metadata from DBMS1. It is assumed that the

`DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE()` procedure was called to grant appropriate privileges to this user at DBMS1.

- User GGADM2 in DBMS2 whose credentials Extract will use to fetch data and metadata from DBMS2. It is assumed that the `DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE()` procedure was called to grant appropriate privileges to this user at DBMS2.
- User GGADM3 in DBMS3 whose credentials Extract will use to fetch data and metadata from DBMS3. It is assumed that the `DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE()` procedure was called to grant appropriate privileges to this user at DBMS3.
- User GGADMCAP in DBMSCAP whose credentials Extract will use to retrieve logical change records from the logmining server at the downstream mining database. It is assumed that the `DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE()` procedure was called to grant appropriate privileges to this user at the downstream mining database DBMSCAP.

This procedure also assumes that the downstream mining database is configured in archive log mode.

In this example, the redo sent by DBMS3 will be mined in real time mode, whereas the redo data sent from DBMS1 and DBMS2 will be mined in archive-log-only mode.

- Prepare the Mining Database to Archive its Local Redo
- Prepare the Mining Database to Accept Redo from the Source Databases
- Prepare the First Source Database to Send Redo to the Mining Database
- Prepare the Second Source Database to Send Redo to the Mining Database
- Prepare the Third Source Database to Send Redo to the Mining Database
- Set up Extracts at Downstream Mining Database

Prepare the Mining Database to Archive its Local Redo

To prepare the mining database to archive its local redo:

1. The downstream mining database must be in archive log mode. You can do this by issuing the following DDL.

```
STARTUP MOUNT;
ALTER DATABASE ARCHIVELOG;
ALTER DATABASE OPEN;
```

2. At the downstream mining database, set `log_archive_dest_1` to archive local redo.

```
ALTER SYSTEM SETLOG_ARCHIVE_DEST_1='LOCATION=/home/arc_dest/
localVALID_FOR=(ONLINE_LOGFILE, PRIMARY_ROLE)'
```

3. Enable `log_archive_dest_1`.

```
ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_1=ENABLE
```

Prepare the Mining Database to Accept Redo from the Source Databases

Because redo data is being accepted in the standby redo logs of the downstream mining database, the appropriate number of correctly sized standby redo logs must exist. If you did not configure the standby logs, see Configuring a Downstream Mining Database .

1. At the downstream mining database, set the second archive log destination in the `LOG_ARCHIVE_DEST_n` initialization parameter as shown in the following example. This is needed to handle archive standby redo logs.

   ```
   ALTER SYSTEM SET
   LOG_ARCHIVE_DEST_2='LOCATION=/home/arc_dest/srl_dbms3
   VALID_FOR=(STANDBY_LOGFILE,PRIMARY_ROLE)'
   ```

2. Enable the `LOG_ARCHIVE_DEST_STATE_2` initialization parameter that corresponds with the `LOG_ARCHIVE_DEST_2` parameter as shown in the following example.

   ```
   ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2=ENABLE
   ```

3. Set `DG_CONFIG` at the downstream mining database to accept redo data from all of the source databases.

   ```
   ALTER SYSTEM SET LOG_ARCHIVE_CONFIG='DG_CONFIG=(dbms1, dbms2, dbms3,
   dbmscap)'
   ```

Prepare the First Source Database to Send Redo to the Mining Database

To prepare the first source database to send redo to the mining database:

1. Make certain that DBMS1 source database is running with the required compatibility.

   ```
   select name, value from v$parameter where name = 'compatible';

   NAME              VALUE
   ---------         ---------------------
   compatible        11.1.0.0.0
   ```

   The minimum compatibility setting required from capture is 11.1.0.0.0.

2. Set `DG_CONFIG` at DBMS1 source database.

   ```
   ALTER SYSTEM SET LOG_ARCHIVE_CONFIG='DG_CONFIG=(dbms1, dbmscap)';
   ```

3. Set up redo transport at DBMS1 source database. The `TEMPLATE` clause is mandatory if you want to send redo data directly to foreign archived logs at the downstream mining database.

   ```
   ALTER SYSTEM
   SET LOG_ARCHIVE_DEST_2='SERVICE=DBMSCAP.EXAMPLE.COM ASYNC OPTIONAL NOREGISTER
   TEMPLATE='/usr/orcl/arc_dest/dbms1/dbms1_arch_%t_%s_%r.log
   VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)DB_UNIQUE_NAME=dbmscap';
   ```

4. Enable the downstream destination.

   ```
   ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2=ENABLE;
   ```

Prepare the Second Source Database to Send Redo to the Mining Database

To prepare the second source database to send redo to the mining database:

1. Make sure that DBMS2 source database is running with the required compatibility.

   ```
   select name, value from v$parameter where name = 'compatible';
   ```

```
NAME                     VALUE
---------                ---------------------
compatible               11.1.0.0.0
```

The minimum compatibility setting required from capture is 11.1.0.0.0.

2. Set `DG_CONFIG` at DBMS2 source database.

```
ALTER SYSTEM SET LOG_ARCHIVE_CONFIG='DG_CONFIG=(dbms2, dbmscap)';
```

3. Set up redo transport at DBMS2 source database. The `TEMPLATE` clause is mandatory if you want to send redo data directly to foreign archived logs at the downstream mining database.

```
ALTER SYSTEM
SET LOG_ARCHIVE_DEST_2='SERVICE=DBMSCAP.EXAMPLE.COM ASYNC OPTIONAL NOREGISTER
TEMPLATE='/usr/orcl/arc_dest/dbms2/dbms2_arch_%t_%s_%r.log
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)DB_UNIQUE_NAME=dbmscap';
```

4. Enable the downstream destination.

```
ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2=ENABLE;
```

Prepare the Third Source Database to Send Redo to the Mining Database

To prepare the third source database to send redo to the mining database:

1. Make sure that DBMS3 source database is running with the required compatibility.

```
select name, value from v$parameter where name = 'compatible';

NAME                     VALUE
---------                ---------------------
compatible               11.1.0.0.0
```

The minimum compatibility setting required from capture is 11.1.0.0.0.

2. Set `DG_CONFIG` at DBMS3 source database.

```
ALTER SYSTEM SET LOG_ARCHIVE_CONFIG='DG_CONFIG=(dbms3, dbmscap)';
```

3. Set up redo transport at DBMS3 source database. Because DBMS3 is the source that will send its online redo logs to the standby redo logs at the downstream mining database, do not specify a `TEMPLATE` clause.

```
ALTER SYSTEM
SET LOG_ARCHIVE_DEST_2='SERVICE=DBMSCAP.EXAMPLE.COM ASYNC OPTIONAL NOREGISTER
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)DB_UNIQUE_NAME=dbmscap';
```

4. Enable the downstream destination.

```
ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2=ENABLE;
```

Set up Extracts at Downstream Mining Database

These steps set up Extract at the downstream database to capture from the archived logs sent by DBMS1 and DBMS2.

- Set up Extract (ext1) to Capture Changes from Archived Logs Sent by DBMS1
- Set up Extract (ext2) to Capture Changes from Archived Logs Sent by DBMS2

- Set up Extract (ext3) to Capture Changes in Real-time Mode from Online Logs Sent by DBMS3

Set up Extract (ext1) to Capture Changes from Archived Logs Sent by DBMS1

Perform the following steps on the DBMSCAP downstream mining database.

1. Register Extract with DBMSCAP for the DBMS1 source database. In the credential store, the alias name of `ggadm1` is linked to a user connect string of `ggadm1@dbms1`.The alias name of `ggadmcap` is linked to a user connect string of `ggadmcap@dbmscap`.

   ```
   GGSCI> DBLOGIN USERIDALIAS ggadm1
   GGSCI> MININGDBLOGIN USERIDALIAS ggadmcap
   GGSCI> REGISTER EXTRACT ext1 DATABASE
   ```

2. Add Extract at the mining database DBMSCAP.

   ```
   GGSCI> ADD EXTRACT ext1 INTEGRATED TRANLOG BEGIN NOW
   ```

3. Edit the Extract parameter file `ext1.prm`. In the credential store, the alias name of `ggadm1` is linked to a user connect string of `ggadm1@dbms1`. The alias name of `ggadmcap` is linked to a user connect string of `ggadmcap@dbmscap`.

   ```
   USERIDALIAS ggadm1
   TRANLOGOPTIONS MININGUSERALIAS ggadmcap
   TRANLOGOPTIONS INTEGRATEDPARAMS (downstream_real_time_mine N)
   ```

4. Start Extract.

   ```
   GGSCI> START EXTRACT ext1
   ```

Set up Extract (ext2) to Capture Changes from Archived Logs Sent by DBMS2

Perform the following steps on the DBMSCAP downstream mining database.

1. Register Extract with the mining database for source database DBMS2. In the credential store, the alias name of `ggadm2` is linked to a user connect string of `ggadm2@dbms2`.The alias name of `ggadmcap` is linked to a user connect string of `ggadmcap@dbmscap`.

   ```
   GGSCI> DBLOGIN USERIDALIAS ggadm2
   GGSCI> MININGDBLOGIN USERIDALIAS ggadmcap
   GGSCI> REGISTER EXTRACT ext2 DATABASE
   ```

2. Create Extract at the mining database.

   ```
   GGSCI> ADD EXTRACT ext2 INTEGRATED TRANLOG, BEGIN NOW
   ```

3. Edit the Extract parameter file `ext2.prm`. In the credential store, the alias name of `ggadm2` is linked to a user connect string of `ggadm2@dbms2`.The alias name of `ggadmcap` is linked to a user connect string of `ggadmcap@dbmscap`.

   ```
   USERIDALIAS ggadm2
   TRANLOGOPTIONS MININGUSERALIAS ggadmcap
   TRANLOGOPTIONS INTEGRATEDPARAMS (downstream_real_time_mine N)
   ```

4. Start Extract.

   ```
   GGSCI> START EXTRACT ext2
   ```

Set up Extract (ext3) to Capture Changes in Real-time Mode from Online Logs Sent by DBMS3

Perform the following steps on the DBMSCAP downstream mining database.

1. Register Extract with the mining database for source database DBMS3. In the credential store, the alias name of `ggadm3` is linked to a user connect string of `ggadm3@dbms3`.The alias name of `ggadmcap` is linked to a user connect string of `ggadmcap@dbmscap`.

   ```
   GGSCI> DBLOGIN USERID ggadm3
   GGSCI> MININGDBLOGIN USERID ggadmcap
   GGSCI> REGISTER EXTRACT ext3 DATABASE
   ```

2. Create Extract at the mining database.

   ```
   GGSCI> ADD EXTRACT ext3 INTEGRATED TRANLOG, BEGIN NOW
   ```

3. Edit the Extract parameter file `ext3.prm`. To enable real-time mining, you must specify `downstream_real_time_mine`. In the credential store, the alias name of `ggadm3` is linked to a user connect string of `ggadm3@dbms3`.The alias name of `ggadmcap` is linked to a user connect string of `ggadmcap@dbmscap`.

   ```
   USERIDALIAS ggadm3
   TRANLOGOPTIONS MININGUSERALIAS ggadmcap
   TRANLOGOPTIONS INTEGRATEDPARAMS (downstream_real_time_mine Y)
   ```

4. Start Extract.

   ```
   GGSCI> START EXTRACT ext3
   ```

Note:

You can create multiple Extracts running in real-time integrated capture mode in the downstream mining database, as long as they all are capturing data from the same source database, such as all capturing for database DBMS3 in the preceding example.

8

Automatic Conflict Detection and Resolution

You can configure Oracle GoldenGate to automatically detect and resolve conflicts that occur when same data is updated concurrently at different sites.

Topics:

- About Automatic Conflict Detection and Resolution
  When Oracle GoldenGate replicates changes between Oracle Databases, you can configure and manage Oracle GoldenGate automatic conflict detection and resolution in these databases. To do this, you must ensure that PL/SQL call is done at the source and the target databases.
- Configuring Automatic Conflict Detection and Resolution
  You can configure Oracle GoldenGate automatic conflict detection and resolution in Oracle Database with the `DBMS_GOLDENGATE_ADM` package.
- Managing Automatic Conflict Detection and Resolution
  You can manage Oracle GoldenGate automatic conflict detection and resolution in Oracle Database with the `DBMS_GOLDENGATE_ADM` package.
- Monitoring Automatic Conflict Detection and Resolution
  You can monitor Oracle GoldenGate automatic conflict detection and resolution in an Oracle Database by querying data dictionary views.

About Automatic Conflict Detection and Resolution

When Oracle GoldenGate replicates changes between Oracle Databases, you can configure and manage Oracle GoldenGate automatic conflict detection and resolution in these databases. To do this, you must ensure that PL/SQL call is done at the source and the target databases.

This feature is intended for use with bi-directional replication.

> **Note:**
>
> This chapter is for the automatic conflict detection and resolution feature that is specific to Oracle GoldenGate 21c (21.1.0) and Oracle Database 21c and later, which is configured in an Oracle Database. There is also a general Oracle GoldenGate feature for conflict detection and resolution, which is called Oracle GoldenGate conflict detection and resolution (CDR). Oracle GoldenGate CDR is configured in the Replicat parameter file.

You can configure only one of the following types of automatic conflict detection and resolution for a single table:

- The automatic conflict detection and resolution feature that is specific to Oracle Database 21c

- Oracle GoldenGate CDR

Automatic Conflict Detection and Resolution

You can configure automatic conflict detection and resolution in an Oracle GoldenGate configuration that replicates tables between Oracle Databases. To configure conflict detection and resolution for a table, call the `ADD_AUTO_CDR` procedure in the `DBMS_GOLDENGATE_ADM` package. These are administration APIs, which are expected to be called by an Oracle GoldenGate administrator who is granted privileges through the `DBMS_GOLDENGATE_AUTH.GRANT_ADMIN_PRIVILEGE`. This user needs to have privileges to modify the affected table. These APIs result in the version number being bumped for the object and also lock the object due to DDL execution.

The administrator user must be logged in to the appropriate PDB when calling these APIs. Three constants, which represent bit flags are now added:

- `EARLIEST_TIMESTAMP_RESOLUTION=0x0001` sets `TOMBSTONE KEY VERSIONING` automatically
- `DELETE_ALWAYS_WINS=0x0002` sets `TOMBSTONE KEY VERSIONING` automatically.
- `IGNORE_SITE_PRIORITY=0x0004`

When Oracle GoldenGate captures changes that originated at an Oracle Database, each change is encapsulated in a row logical change record (LCR). A row LCR is a structured representation of a DML row change. Each row LCR includes the operation type, old column values, and new column values. Multiple row LCRs can be part of a single database transaction.

When more than one replica of a table allows changes to the table, a conflict can occur when a change is made to the same row in two different databases at nearly the same time. Oracle GoldenGate replicates changes using the row LCRs. It detects a conflict by comparing the old values in the row LCR for the initial change from the origin database with the current values of the corresponding table row at the destination database identified by the key columns. If any column value does not match, then there is a conflict.

After a conflict is detected, Oracle GoldenGate can resolve the conflict by overwriting values in the row with some values from the row LCR, ignoring the values in the row LCR, or computing a delta to update the row values.

Automatic conflict detection and resolution does not require application changes for the following reasons:

- Oracle Database automatically creates and maintains invisible timestamp columns.
- Inserts, updates, and deletes use the delete tombstone log table to determine if a row was deleted.
- LOB column conflicts can be detected.
- Oracle Database automatically configures supplemental logging on required columns.

See Also:

- *Oracle Database Utilities* for information about supplemental logging

Requirements for Automatic Conflict Detection and Resolution

Learn the requirements for automatic conflict detection and resolution (ACDR).

Supplemental logging is required to ensure that each row LCR has the information required to detect and resolve a conflict. Supplemental logging places additional information in the redo log for the columns of a table when a DML operation is performed on the table. When you configure a table for Oracle GoldenGate conflict detection and resolution, supplemental logging is configured automatically for all of the columns in the table. The additional information in the redo log is placed in an LCR when a table change is replicated.

Extract must be used for capturing. Integrated Replicat or parallel Replicat in integrated mode must be used on the apply side. Replicat needs to add the parameter `MAPINVISIBLECOLUMNS`. `LOGALLSUPCOLS` should remain the default.

There is a hidden field `KEYVER$$` of type timestamp that is optionally added to the `DELETE TOMBSTONE` table. This field is required for `EARLIEST TIMESTAMP`, `DELETE ALWAYS WINS`, and `SITE PRIORITY` resolution and it also exists in the base table. The existence of the field in the base table needs to be provided in the trail file metadata as a flag or token.

Primary Key updates is also supported in the `DELETE TOMBSTONE` table. An entry is inserted into the `DELETE TOMBSTONE` table for the row of the original key value (before image). The logic in the Extract which matches inserts in the `DELETE TOMBSTONE` table to deletes also needs to be matched to PK updates, or unique key (UK) with at least one non-nullable field, if there is no PK.

Site priority needs support from the Replicat, both the parameters are implemented and the setting is passed to the apply.

- Compatibility and Migration

Compatibility and Migration

Replicating from a base table which doesn’t have a `KEYVER$$` to a target, which has `EARLIEST TIMESTAMP` resolution support, `DELETE ALWAYS WINS` resolution, or SITE PRIORITY, will receive an error in cases involving `DELETE` or PK Update.

Replicating from a base table, which has a `KEYVER$$` to a target, which does not, will ignore the `KEYVER$$`, when replicating to an earlier release, then the field is dropped or is not supported).

Example

The following table shows `EARLIEST TIMESTAMP` resolution on **Site1** with no `keyver$$` column or earlier RDBMS version replication to **Site 2**

Site 1	Site 2	Description
insert	insert	If the **Site1** `CDRTS$` timestamp is earlier then **Site1** wins else **Site2** wins
insert	update	Same as insert insert.
insert	delete	Conflict cannot be resolved, depending on configuration, error goes to error queue, discard and so on.
insert	pkupdate	Same as insert delete
update	insert	Same as insert insert
update	update	Same as insert insert
update	delete	Same as insert delete
update	pkupdate	Same as insert delete
delete	insert	Same as insert delete
delete	update	Same as insert delete
pkupdate	insert	Same as insert delete
pkupdate	update	Same as insert delete
pkupdate	delete	Same as insert delete

Primary Key Updates (pkupdate) interoperability will not resolve correctly without a backport. Besides the interoperability problems listed above, pkupdates that are replicated to earlier versions of the RDBMS, will not resolve correctly. A conflicting insert and replicated pkupdate on the earlier RDBMS may result in 2 rows. The insert will succeed to the original row and the pkupdate will succeed to update or create the new row.

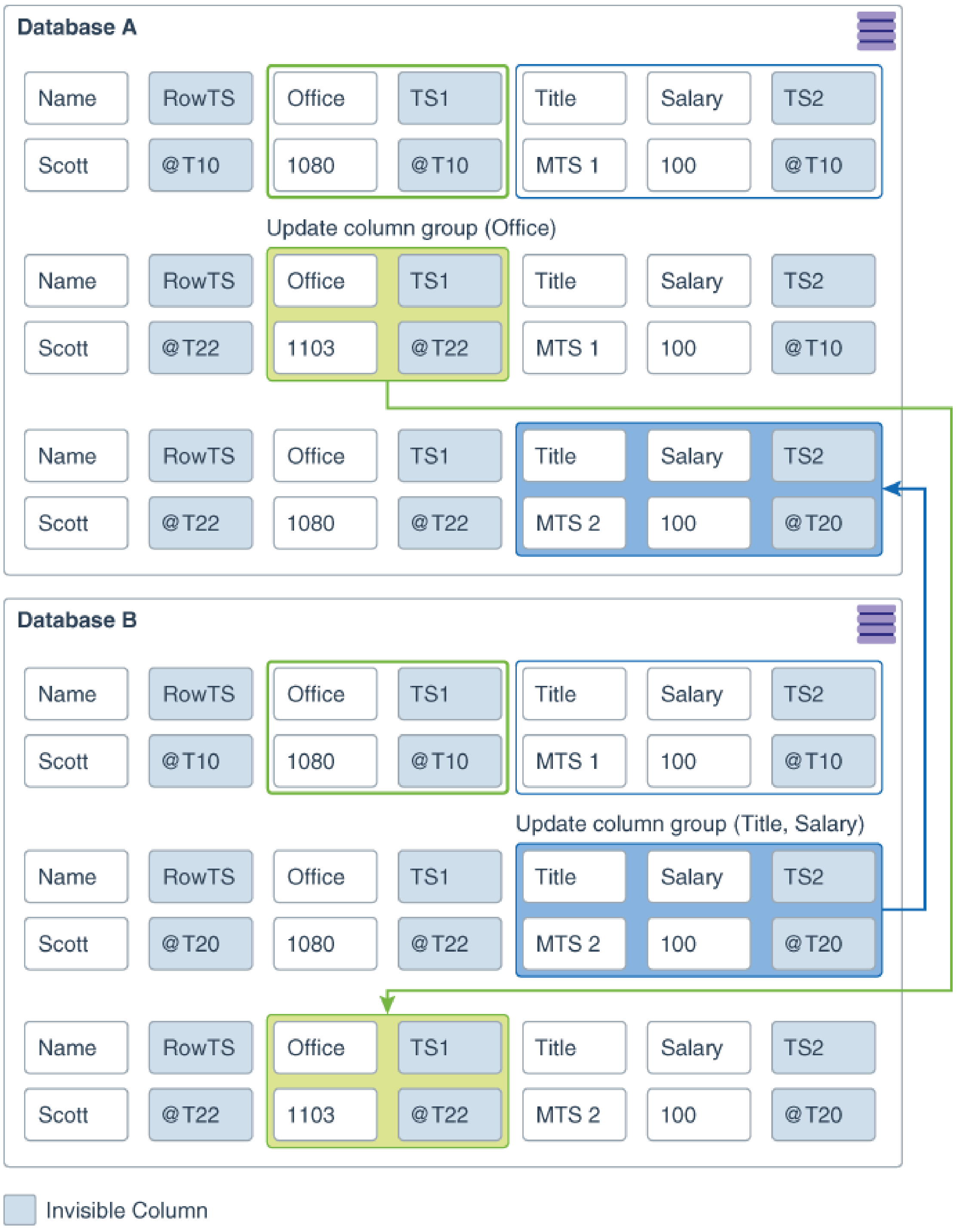
No upgrage or downgrade scripts are needed because the changes are just to procedure attributes and view columns.

Column Groups

A column group is a logical grouping of one or more columns in a replicated table. When you add a column group, conflict detection and resolution is performed on the columns in the column group separately from the other columns in the table.

When you configure a table for Oracle GoldenGate conflict detection and resolution with the `ADD_AUTO_CDR` procedure, all of the scalar columns in the table are added to a default column group. To define other column groups for the table, run the `ADD_AUTO_CDR_COLUMN_GROUP` procedure. Any columns in the table that are not part of a user-defined column group remain in the default column group for the table.

Column groups enable different databases to update different columns in the same row at nearly the same time without causing a conflict. When column groups are configured for a table, conflicts can be avoided even if different databases update the same row in the table. A conflict is not detected if the updates change the values of columns in different column groups.



This example shows a row being replicated at database A and database B. The following two column groups are configured for the replicated table at each database:

- One column group includes the `Office` column. The invisible timestamp column for this column group is `TS1`.
- Another column group includes the `Title` and `Salary` columns. The invisible timestamp column for this column group is `TS2`.

These column groups enable database A and database B to update the same row at nearly the same time without causing a conflict. Specifically, the following changes are made:

- At database A, the value of `Office` was changed from `1080` to `1030`.
- At database B, the value of `Title` was changed from `MTS1` to `MTS2`.

Because the `Office` column and the `Title` column are in different column groups, the changes are replicated without a conflict being detected. The result is that values in the row are same at both databases after each change has been replicated.

Piecewise LOB Updates

A set of lob operations composed of `LOB WRITE`, `LOB ERASE`, and `LOB TRIM` is a piecewise LOB update. When a table that contains LOB columns is configured for conflict detection and resolution, each LOB column is placed in its own column group, and the column group has its own hidden timestamp column. The timestamp column is updated on the first piecewise LOB operation.

For a LOB column, a conflict is detected and resolved in the following ways:

- If the timestamp for the LOB's column group is later than the corresponding LOB column group in the row, then the piecewise LOB update is applied.
- If the timestamp for the LOB's column group is earlier than the corresponding LOB column group in the row, then the LOB in the table row is retained.
- If the row does not exist in the table, then an error occurs

DELETE TOMBSTONE Table

`DELETE TOMBSTONE` table is a marker for a deleted record to distinguish it from a record, which never existed. A `DELETE TOMBSTONE` table contains at minimum the key columns and operation timestamp. This information is required for delete convergence becasue some incoming updates and inserts may be delayed from another site and the incoming LCR needs to be filtered against the tombstone operation timestamp to determine whether it should be applied.

Earliest Timestamp Conflict Detection and Resolution

Columns with names of the form `CDRTS$` *`column group`* and `CDRTS$ROW` are used to contain timestamps that reflect modification times for column groups and the row. The `DBMS_GOLDENGATE_ADM` procedures `ADD_AUTO_CDR()`, `ADD_AUTO_CDR_COLUMN_GROUP()`, `REMOVE_AUTO_CDR()`, `REMOVE_AUTO_CDR_COLUMN_GROUP()`, `ALTER_AUTO_CDR()`, and `ALTER_AUTO_CDR_COLUMN_GROUP()` are presently used to configure ACDR with latest timestamp resolution. They are also used in configuration of ACDR with earliest timestamp resolution, The field `ADDITIONAL_OPTIONS` in both `ADD_AUTO_CDR()` and `ALTER_AUTO_CDR()` turn on the use of earliest timestamp. Turning on earliest timestamp automatically turn on versioning, which adds a new hidden column

`KEYVER$$` (version number) of type timestamp. A new flag value is added to `acdrflags_kqldtvc` to indicate earliest timestamp usage. This field is also added to the `DELETE TOMBSTONE` table. Delete conflicts are the reason that version number is needed. With an earliest timestamp resolution, delete conflicts, which can be transparent, might not only incorrectly succeed, they might prevent new inserts of the row (new versions). With a version timestamp, the delete can be correctly resolved against a row DML for the same row version.

The original insert of the row receives the current timestamp from its default value. The delete of this row then inserts the version number and the time when this row was inserted, into the tombstone table when there is a delete. On a new insert, by default, the version number receives the current timestamp again, thereby avoiding a false conflict with the present delete entries in the tombstone table.

Example

For key version `kv` and timestamp `ts`

Database 1: `insert tab1 key1 kv1 ts1`

Database 2: `delete tab1 key1 kv1 ts1`

Insertion to DELETE TOMBSTONE table `key1 kv1 ts1`

Database 1: `insert tab1 key1 kv2 ts2`

Without using the key version, the insert would be ignoreed, the delete timestamp is earlier. As the key version is used, you know that `kv2` is not the version of the row that was deleted and the insert succeeds.

Latest Timestamp Conflict Detection and Resolution

When you run the `ADD_AUTO_CDR` procedure in the `DBMS_GOLDENGATE_ADM` package to configure a table for automatic Oracle GoldenGate conflict detection and resolution, a hidden timestamp column is added to the table. This hidden timestamp column records the time of a row change, and this information is used to detect and resolve conflicts.

When a row LCR is applied, a conflict can occur for an `INSERT`, `UPDATE`, or `DELETE` operation. The following table describes each type of conflict and how it is resolved.

Operation	Conflict Detection	Conflict Resolution
`INSERT`	A conflict is detected when the table has the same value for a key column as the new value in the row LCR.	If the timestamp of the row LCR is later than the timestamp in the table row, then the values in the row LCR replace the values in the table. If the timestamp of the row LCR is earlier than the timestamp in the table row, then the row LCR is discarded, and the table values are retained.

Operation	Conflict Detection	Conflict Resolution
`UPDATE`	A conflict is detected in each of the following cases: • There is a mismatch between the timestamp value in the row LCR and the timestamp value of the corresponding row in the table. • There is a mismatch between an old value in a column group in the row LCR does not match the column value in the corresponding table row. A column group is a logical grouping of one or more columns in a replicated table. • The table row does not exist. If the row is in the tombstone table, then this is referred to as an update-delete conflict.	If there is a value mismatch and the timestamp of the row LCR is later than the timestamp in the table row, then the values in the row LCR replace the values in the table. If there is a value mismatch and the timestamp of the row LCR is earlier than the timestamp in the table row, then the row LCR is discarded, and the table values are retained. If the table row does not exist and the timestamp of the row LCR is later than the timestamp in the tombstone table row, then the row LCR is converted from an `UPDATE` operation to an `INSERT` operation and inserted into the table. If the table row does not exist and the timestamp of the row LCR is earlier than the timestamp in the tombstone table row, then the row LCR is discarded. If the table row does not exist and there is no corresponding row in the tombstone table, then the row LCR is converted from an `UPDATE` operation to an `INSERT` operation and inserted into the table.
`DELETE`	A conflict is detected in each of the following cases: • There is a mismatch between the timestamp value in the row LCR and the timestamp value of the corresponding row in the table. • The table row does not exist.	If the timestamp of the row LCR is later than the timestamp in the table, then delete the row from the table. If the timestamp of the row LCR is earlier than the timestamp in the table, then the row LCR is discarded, and the table values are retained. If the delete is successful, then log the row LCR by inserting it into the tombstone table. If the table row does not exist, then log the row LCR by inserting it into the tombstone table.

Delete Always Wins Timestamp CDR

`DELETE ALWAYS WINS` is enabled through the field `ADDITIONAL_OPTIONS` in both `DBMS_GOLDENGATE_ADM` procedures `ADD_AUTO_CDR()` and `ALTER_AUTO_CDR()`. This is again a delete conflict resolution method, which is not using latest timestamp resolution, therefore, versioning is needed. Turning on `DELETE ALWAYS WINS` automatically turns on versioning, which adds a new hidden column `KEYVER$$` (version number) of type timestamp. A new flag value is also added to `acdrflags_kqldtvc` to indicate `DELETE ALWAYS WINS` usage. This field is also added to the `DELETE TOMBSTONE` table. The same versioning issues exist as the EARLIEST TIMESTAMP resolution.

Exmaple:

Key Version `kv` and Timestamp `ts`

Database 1: `insert tab1 key1 kv1 ts1`

Database 2: `delete tab1 key1 kv1 ts1`

Insertion to `DELETE TOMBSTONE` table `key1 kv1 ts1`

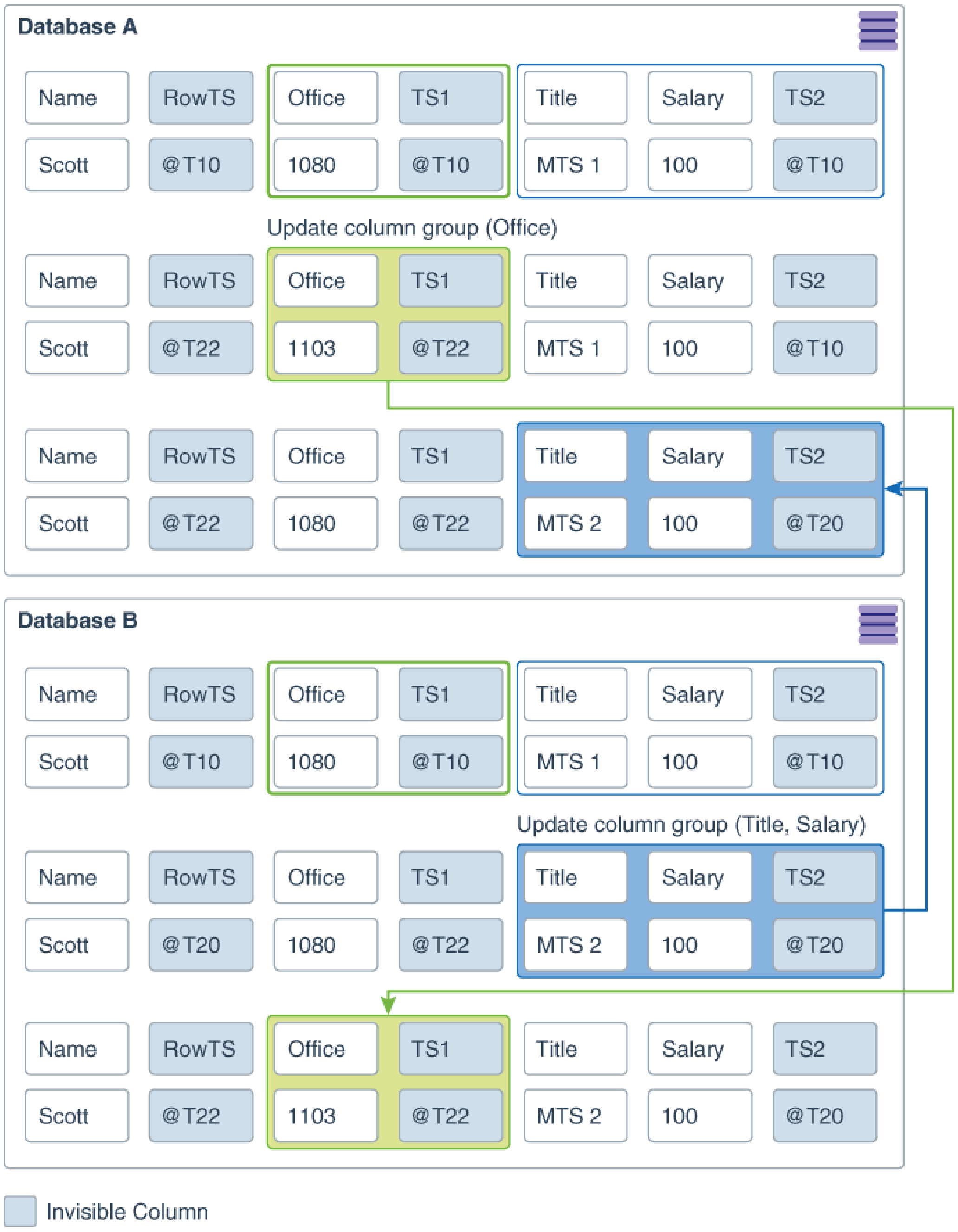
Database 1: `insert tab1 key1 kv2 ts2`

Without using the key version, the insert would be ignored, the delete always wins. As the key version is used, you know that `kv2` is not the version of the row that was deleted and the insert succeeds.

Delta Conflict Detection and Resolution

With delta conflict detection, a conflict occurs when a value in the old column list of the row LCR differs from the value for the corresponding row in the table.

To configure delta conflict detection and resolution for a table, run the `ADD_AUTO_CDR_DELTA_RES` procedure in the `DBMS_GOLDENGATE_ADM` package. The delta resolution method does not depend on a timestamp or an extra resolution column. With delta conflict resolution, the conflict is resolved by adding the difference between the new and old values in the row LCR to the value in the table. This resolution method is generally used for financial data such as an account balance. For example, if a bank balance is updated at two sites concurrently, then the converged value accounts for all debits and credits.



This example shows a row being replicated at database A and database B. The `Balance` column is designated as the column on which delta conflict resolution is performed, and the `TS1` column is the invisible timestamp column to track the time of each change to the `Balance` column. A change is made to the `Balance` value in the row in both databases at nearly the same time (`@T20` in database A and `@T22` in database B). These changes result in a conflict, and delta conflict resolution is used to resolve the conflict in the following way:

- At database A, the value of `Balance` was changed from `100` to `110`. Therefore, the value was increased by 10.

- At database B, the value of Balance was changed from 100 to 120. Therefore, the value was increased by 20.
- To resolve the conflict at database A, the value of the difference between the new and old values in the row LCR to the value in the table. The difference between the new and old values in the LCR is 20 (120–100=20). Therefore, the current value in the table (110) is increased by 20 so that the value after conflict resolution is 130.
- To resolve the conflict at database B, the value of the difference between the new and old values in the row LCR to the value in the table. The difference between the new and old values in the LCR is 10 (110–100=10). Therefore, the current value in the table (120) is increased by 10 so that the value after conflict resolution is 130.

After delta conflict resolution, the value of the Balance column is the same for the row at database A and database B.

Site Priority CDR

Note:

SITE PRIORITY resolution takes precedence over all COLUMN GROUP resolution settings.

Note:

If SITE PRIORITY Replicat parameter is not placed before applicable map statements in the parameter file, it will not work. This parameter must be placed before the applicable map statements.

Priority resolution specified in Replicat parameter file between source and target in conflict resolution.

SITE PRIORITY is turned on for a database or PDB in the Replicat parameter file with the parameter ACDR SITE_PRIORITY {source_db_name}{OVERWRITE | IGNORE } specified to turn on SITE PRIORITY resolution for a table. If OVERWRITE is specified, then the source table has priority and conflicts are resolved by OVERWRITE. Conversely, if IGNORE is specified, then the target table has priority and source table change are ignored in a conflict. SITE PRIORITY resolution can be turned off by the field ADDITIONAL_OPTIONS in both DBMS_GOLDENGATE_ADM procedure ADD_AUTO_CDR() and ALTER_AUTO_CDR() setting the bit IGNORE_SITE_PRIORITY. Every Replicat source-target relationship can be set up differently, therefore, convergence is dependent on user setup.

Track PK Updates in Delete Tombstone

Full support of primary key (PK) updates requires handling conflicts on both the rows represented by the before image of the key and the row represented by the after image of the key. A PK update is an autonomous delete and insert, so, the PK update conflicts must be supported as a delete for conflicts with the before image of the key and inserts with the after image of the key (and row).

Supporting the PK update as a delete of the row represented by the before image of the key means that it should insert into the delete tombstone table as a delete. An update internal trigger is added to insert into the tombstone table when the PK is updated (actually the row identifying key, either the PK if it exists or the chosen UK with at least one non-nullable column). As a PK update may lead to two conflicts, up to two resolutions are attempted at the row level, delete of the row with the original PK and the insert of the row with the new PK.

Example: Using latest timestamp resolution

Database 1: Update to `tab1 key1 at ts1`

Database 2: Update to `tab1 key1 set key1 to key2 ts2`

Database 3: Update to `tab1 key2 ts3`

In this scenario, it appears that at the row level `tab1` row with `key1` should be deleted and the database 3 update should be the final modification of `tab1` row key2. If instead the database 2 is at `ts3` and database 3 is at `ts3`, then the PK update at database 2 would be the final modification of `tab1` row `key2`.

Now, consider a case where the database 1 was at `ts3`, database 2 at `ts2` and database 3 at `ts1`, then the update to `tab1` row `key1` on database 1 should succeed and the PK update from database 2 on `tab1` row `key2` should succeed. At this point, it looks like the complete resolution is that both the delete at the before image and the insert at the after image must be resolved separately. This implies that they are not dependent on each other and a loss for one, is not a loss for both.

Configuring Automatic Conflict Detection and Resolution

You can configure Oracle GoldenGate automatic conflict detection and resolution in Oracle Database with the `DBMS_GOLDENGATE_ADM` package.

For the Replicat parameter file you need to add a `MAP` statement that includes the table to be replicated and the `MAPINVISIBLECOLUMNS` parameter.

- Configuring Latest Timestamp Conflict Detection and Resolution
  The `ADD_AUTO_CDR` procedure in the `DBMS_GOLDENGATE_ADM` package configures latest timestamp conflict detection and resolution. The `ADD_AUTO_CDR_COLUMN_GROUP` procedure adds optional column groups.
- Configuring Delta Conflict Detection and Resolution
  The `ADD_AUTO_CDR_DELTA_RES` procedure in the `DBMS_GOLDENGATE_ADM` package configures delta conflict detection and resolution.

Configuring Latest Timestamp Conflict Detection and Resolution

The `ADD_AUTO_CDR` procedure in the `DBMS_GOLDENGATE_ADM` package configures latest timestamp conflict detection and resolution. The `ADD_AUTO_CDR_COLUMN_GROUP` procedure adds optional column groups.

With latest timestamp conflict detection and resolution, a conflict is detected when the timestamp column of the row LCR does not match the timestamp of the corresponding table row. The row LCR is applied if its timestamp is later. Otherwise, the row LCR is discarded, and the table row is not changed. When you run the `ADD_AUTO_CDR` procedure, it adds an invisible timestamp column for each row in the specified table and configures timestamp conflict detection and resolution. When you use the `ADD_AUTO_CDR_COLUMN_GROUP` procedure to add one or more column groups, it adds a timestamp for the column group and configures timestamp conflict detection and resolution for the column group.

You can configure an Oracle GoldenGate administrator using the `GRANT_ADMIN_PRIVILEGE` procedure in the `DBMS_GOLDENGATE_ADM` package.

1. Connect to the inbound server database as a Oracle GoldenGate administrator.
2. Run the `ADD_AUTO_CDR` procedure and specify the table to configure for latest timestamp conflict detection and resolution.
3. Optional: Run the `ADD_AUTO_CDR_COLUMN_GROUP` procedure and specify one or more column groups in the table.
4. Repeat the previous steps in each Oracle Database that replicates the table.

Example 8-1 Configuring the Latest Timestamp Conflict Detection and Resolution for a Table

This example configures latest timestamp conflict detection and resolution for the `hr.employees` table.

```
BEGIN
  DBMS_GOLDENGATE_ADM.ADD_AUTO_CDR(
    schema_name => 'hr',
    table_name  => 'employees');
END;
/
```

Example 8-2 Configuring Column Groups

This example configures the following column groups for timestamp conflict resolution on the `hr.employees` table:

- The `job_identifier_cg` column group includes the `job_id`, `department_id`, and `manager_id` columns.
- The `compensation_cg` column group includes the `salary` and `commission_pct` columns.

```
BEGIN
  DBMS_GOLDENGATE_ADM.ADD_AUTO_CDR_COLUMN_GROUP(
    schema_name       => 'hr',
    table_name        => 'employees',
    column_list       => 'job_id,department_id,manager_id',
    column_group_name => 'job_identifier_cg');
END;
/

BEGIN
  DBMS_GOLDENGATE_ADM.ADD_AUTO_CDR_COLUMN_GROUP(
    schema_name       => 'hr',
    table_name        => 'employees',
    column_list       => 'salary,commission_pct',
    column_group_name => 'compensation_cg');
END;
/
```

Configuring Delta Conflict Detection and Resolution

The ADD_AUTO_CDR_DELTA_RES procedure in the DBMS_GOLDENGATE_ADM package configures delta conflict detection and resolution.

With delta conflict resolution, you specify one column for which conflicts are detected and resolved. The conflict is detected if the value of the column in the row LCR does not match the corresponding value in the table. The conflict is resolved by adding the difference between the new and old values in the row LCR to the value in the table.

You can configure an Oracle GoldenGate administrator using the GRANT_ADMIN_PRIVILEGE procedure in the DBMS_GOLDENGATE_ADM package.

1. Connect to the inbound server database as an Oracle GoldenGate administrator.
2. Run the ADD_AUTO_CDR procedure and specify the table to configure for latest timestamp conflict detection and resolution.
3. Run the ADD_AUTO_CDR_DELTA_RES procedure and specify the column on which delta conflict detection and resolution is performed.
4. Repeat the previous steps in each Oracle Database that replicates the table.

Example 8-3 Configuring Delta Conflict Detection and Resolution for a Table

This example configures delta conflict detection and resolution for the order_total column in the oe.orders table.

```
BEGIN
  DBMS_GOLDENGATE_ADM.ADD_AUTO_CDR(
    schema_name => 'oe',
    table_name  => 'orders');
END;
/

BEGIN
  DBMS_GOLDENGATE_ADM.ADD_AUTO_CDR_DELTA_RES(
    schema_name => 'oe',
    table_name  => 'orders',
    column_name => 'order_total');
END;
/
```

Managing Automatic Conflict Detection and Resolution

You can manage Oracle GoldenGate automatic conflict detection and resolution in Oracle Database with the DBMS_GOLDENGATE_ADM package.

- Altering Conflict Detection and Resolution for a Table
- Altering a Column Group
- Purging Tombstone Rows
- Removing Conflict Detection and Resolution From a Table
- Removing a Column Group

- Removing Delta Conflict Detection and Resolution

Altering Conflict Detection and Resolution for a Table

The `ALTER_AUTO_CDR` procedure in the `DBMS_GOLDENGATE_ADM` package alters conflict detection and resolution for a table.

Oracle GoldenGate automatic conflict detection and resolution must be configured for the table:

1. Connect to the inbound server database as the Oracle GoldenGate administrator.
2. Run the `ALTER_AUTO_CDR` procedure and specify the table to configure for latest timestamp conflict detection and resolution.
3. Repeat all of the previous steps in each Oracle Database that replicates the table.

Example 8-4 Altering Conflict Detection and Resolution for a Table

This example alters conflict detection and resolution for the `hr.employees` table to specify that delete conflicts are tracked in a tombstone table.

```
BEGIN
  DBMS_GOLDENGATE_ADM.ALTER_AUTO_CDR(
    schema_name       => 'hr',
    table_name        => 'employees',
    tombstone_deletes => TRUE);
END;
/
```

Altering a Column Group

The `ALTER_AUTO_CDR_COLUMN_GROUP` procedure alters a column group.

1. Connect to the inbound server database as an Oracle GoldenGate administrator.
2. Run the `ALTER_AUTO_CDR_COLUMN_GROUP` procedure and specify one or more column groups in the table.
3. Repeat all of the previous steps in each Oracle Database that replicates the table.

Example 8-5 Altering a Column Group

This example removes the `manager_id` column from the `job_identifier_cg` column group for the `hr.employees` table.

```
BEGIN
  DBMS_GOLDENGATE_ADM.ALTER_AUTO_CDR_COLUMN_GROUP(
    schema_name        => 'hr',
    table_name         => 'employees',
    column_group_name  => 'job_identifier_cg',
    remove_column_list => 'manager_id');
END;
/
```

Note:

If there is more than one column, then use a comma-separated list.

Purging Tombstone Rows

The PURGE_TOMBSTONES procedure removes tombstone rows that were recorded before a specified date and time. This procedure removes the tombstone rows for all tables configured for conflict resolution in the database.

It might be necessary to purge tombstone rows periodically to keep the tombstone log from growing too large over time.

1. Connect to the inbound server database as an Oracle GoldenGate administrator.
2. Run the PURGE_TOMBSTONES procedure and specify the date and time.

Example 8-6 Purging Tombstone Rows

This example purges all tombstone rows recorded before 3:00 p.m. on December, 1, 2015 Eastern Standard Time. The timestamp must be entered in TIMESTAMP WITH TIME ZONE format.

```
EXEC DBMS_GOLDENGATE_ADM.PURGE_TOMBSTONES('2015-12-01 15:00:00.000000
EST');
```

Removing Conflict Detection and Resolution From a Table

The REMOVE_AUTO_CDR procedure in the DBMS_GOLDENGATE_ADM package removes automatic conflict detection and resolution from a table. This procedure also removes any column groups and delta conflict detection and resolution configured for the table.

1. Connect to the inbound server database as an Oracle GoldenGate administrator.
2. Run the REMOVE_AUTO_CDR procedure and specify the table.
3. Repeat all of the previous steps in each Oracle Database that replicates the table.

Example 8-7 Removing Conflict Detection and Resolution for a Table

This example removes conflict detection and resolution for the hr.employees table.

```
BEGIN
  DBMS_GOLDENGATE_ADM.REMOVE_AUTO_CDR(
    schema_name => 'hr',
    table_name  => 'employees');
END;
/
```

Removing a Column Group

The REMOVE_AUTO_CDR_COLUMN_GROUP procedure removes a column group.

1. Connect to the inbound server database as an Oracle GoldenGate administrator.

2. Run the REMOVE_AUTO_CDR_COLUMN_GROUP procedure and specify the name of the column group.
3. Repeat all of the previous steps in each Oracle Database that replicates the table.

Example 8-8 Removing a Column Group

This example removes the compensation_cg column group from the hr.employees table.

```
BEGIN
  DBMS_GOLDENGATE_ADM.REMOVE_AUTO_CDR_COLUMN_GROUP(
    schema_name       => 'hr',
    table_name        => 'employees',
    column_group_name => 'compensation_cg');
END;
/
```

Removing Delta Conflict Detection and Resolution

The REMOVE_AUTO_CDR_DELTA_RES procedure in the DBMS_GOLDENGATE_ADM package removes delta conflict detection and resolution for a column.

Delta conflict detection and resolution must be configured for the specified column.

1. Connect to the inbound server database as an Oracle GoldenGate administrator.
2. Run the REMOVE_AUTO_CDR_DELTA_RES procedure and specify the column.
3. Repeat all of the previous steps in each Oracle Database that replicates the table.

Example 8-9 Removing Delta Conflict Detection and Resolution for a Table

This example removes delta conflict detection and resolution for the order_total column in the oe.orders table.

```
BEGIN
  DBMS_GOLDENGATE_ADM.REMOVE_AUTO_CDR_DELTA_RES(
    schema_name => 'oe',
    table_name  => 'orders',
    column_name => 'order_total');
END;
/
```

Monitoring Automatic Conflict Detection and Resolution

You can monitor Oracle GoldenGate automatic conflict detection and resolution in an Oracle Database by querying data dictionary views.

- Displaying Information About the Tables Configured for Conflicts
- Displaying Information About Conflict Resolution Columns
- Displaying Information About Column Groups

Displaying Information About the Tables Configured for Conflicts

The `ALL_GG_AUTO_CDR_TABLES` view displays information about the tables configured for Oracle GoldenGate automatic conflict detection and resolution.

1. Connect to the database.
2. Query the `ALL_GG_AUTO_CDR_TABLES` view.

Example 8-10 Displaying Information About the Tables Configured for Conflict Detection and Resolution

This query displays the following information about the tables that are configured for conflict detection and resolution:

- The table owner for each table.
- The table name for each table.
- The tombstone table used to store rows deleted for update-delete conflicts, if a tombstone table is configured for the table.
- The hidden timestamp column used for conflict resolution for each table.

```
COLUMN TABLE_OWNER FORMAT A15
COLUMN TABLE_NAME FORMAT A15
COLUMN TOMBSTONE_TABLE FORMAT A15
COLUMN ROW_RESOLUTION_COLUMN FORMAT A25

SELECT TABLE_OWNER,
       TABLE_NAME,
       TOMBSTONE_TABLE,
       ROW_RESOLUTION_COLUMN
  FROM ALL_GG_AUTO_CDR_TABLES
  ORDER BY TABLE_OWNER, TABLE_NAME;
```

Your output looks similar to the following:

```
TABLE_OWNER     TABLE_NAME      TOMBSTONE_TABLE ROW_RESOLUTION_COLUMN
--------------- --------------- ---------------
-------------------------
HR              EMPLOYEES       DT$_EMPLOYEES   CDRTS$ROW
OE              ORDERS          DT$_ORDERS      CDRTS$ROW
```

Displaying Information About Conflict Resolution Columns

The `ALL_GG_AUTO_CDR_COLUMNS` view displays information about the columns configured for Oracle GoldenGate automatic conflict detection and resolution.

The columns can be configured for row or column automatic conflict detection and resolution. The columns can be configured for latest timestamp conflict resolution in a column group. In addition, a column can be configured for delta conflict resolution.

1. Connect to the database as an Oracle GoldenGate administrator.
2. Query the `ALL_GG_AUTO_CDR_COLUMNS` view.

Example 8-11 Displaying Information About Column Groups

This query displays the following information about the tables that are configured for conflict detection and resolution:

- The table owner for each table.
- The table name for each table.
- If the column is in a column group, then the name of the column group.
- The column name.
- If the column is configured for latest timestamp conflict resolution, then the name of the hidden timestamp column for the column.

```
COLUMN TABLE_OWNER FORMAT A10
COLUMN TABLE_NAME FORMAT A10
COLUMN COLUMN_GROUP_NAME FORMAT A17
COLUMN COLUMN_NAME FORMAT A15
COLUMN RESOLUTION_COLUMN FORMAT A23

SELECT TABLE_OWNER,
       TABLE_NAME,
       COLUMN_GROUP_NAME,
       COLUMN_NAME,
       RESOLUTION_COLUMN
  FROM ALL_GG_AUTO_CDR_COLUMNS
  ORDER BY TABLE_OWNER, TABLE_NAME;
```

Your output looks similar to the following:

```
TABLE_OWNE TABLE_NAME COLUMN_GROUP_NAME COLUMN_NAME     RESOLUTION_COLUMN
---------- ---------- ----------------- ---------------
-----------------------
HR         EMPLOYEES  COMPENSATION_CG   COMMISSION_PCT  CDRTS$COMPENSATION_CG
HR         EMPLOYEES  COMPENSATION_CG   SALARY          CDRTS$COMPENSATION_CG
HR         EMPLOYEES  JOB_IDENTIFIER_CG MANAGER_ID
CDRTS$JOB_IDENTIFIER_CG
HR         EMPLOYEES  JOB_IDENTIFIER_CG JOB_ID
CDRTS$JOB_IDENTIFIER_CG
HR         EMPLOYEES  JOB_IDENTIFIER_CG DEPARTMENT_ID
CDRTS$JOB_IDENTIFIER_CG
HR         EMPLOYEES  IMPLICIT_COLUMNS$ PHONE_NUMBER    CDRTS$ROW
HR         EMPLOYEES  IMPLICIT_COLUMNS$ LAST_NAME       CDRTS$ROW
HR         EMPLOYEES  IMPLICIT_COLUMNS$ HIRE_DATE       CDRTS$ROW
HR         EMPLOYEES  IMPLICIT_COLUMNS$ FIRST_NAME      CDRTS$ROW
HR         EMPLOYEES  IMPLICIT_COLUMNS$ EMAIL           CDRTS$ROW
HR         EMPLOYEES  IMPLICIT_COLUMNS$ EMPLOYEE_ID     CDRTS$ROW
OE         ORDERS     IMPLICIT_COLUMNS$ ORDER_MODE      CDRTS$ROW
OE         ORDERS     IMPLICIT_COLUMNS$ ORDER_ID        CDRTS$ROW
OE         ORDERS     IMPLICIT_COLUMNS$ ORDER_DATE      CDRTS$ROW
OE         ORDERS     IMPLICIT_COLUMNS$ CUSTOMER_ID     CDRTS$ROW
OE         ORDERS     DELTA$            ORDER_TOTAL
OE         ORDERS     IMPLICIT_COLUMNS$ PROMOTION_ID    CDRTS$ROW
OE          ORDERS      IMPLICIT_COLUMNS$ ORDER_STATUS     CDRTS$ROW
OE          ORDERS      IMPLICIT_COLUMNS$ SALES_REP_ID     CDRTS$ROW
```

In this example, the columns with `IMPLICIT_COLUMNS$` for the column group name are configured for row conflict detection and resolution, but they are not part of a column group. The columns with `DELTA$` for the column group name are configured for delta conflict detection and resolution, and these columns do not have a resolution column.

Displaying Information About Column Groups

The `ALL_GG_AUTO_CDR_COLUMN_GROUPS` view displays information about the column groups configured for Oracle GoldenGate automatic conflict detection and resolution.

You can configure Oracle GoldenGate automatic conflict detection and resolution using the `ADD_AUTO_CDR` procedure in the `DBMS_GOLDENGATE_ADM` package. You can configure column groups using the `ADD_AUTO_CDR_COLUMN_GROUP` procedure in the `DBMS_GOLDENGATE_ADM` package.

1. Connect to the database as an Oracle GoldenGate administrator.
2. Query the `ALL_GG_AUTO_CDR_COLUMN_GROUPS` view.

Example 8-12 Displaying Information About Column Groups

This query displays the following information about the tables that are configured for conflict detection and resolution:

- The table owner.
- The table name.
- The name of the column group.
- The hidden timestamp column used for conflict resolution for each column group.

```
COLUMN TABLE_OWNER FORMAT A15
COLUMN TABLE_NAME FORMAT A15
COLUMN COLUMN_GROUP_NAME FORMAT A20
COLUMN RESOLUTION_COLUMN FORMAT A25

SELECT TABLE_OWNER,
       TABLE_NAME,
       COLUMN_GROUP_NAME,
       RESOLUTION_COLUMN
  FROM ALL_GG_AUTO_CDR_COLUMN_GROUPS
  ORDER BY TABLE_OWNER, TABLE_NAME;
```

The output looks similar to the following:

```
TABLE_OWNER     TABLE_NAME      COLUMN_GROUP_NAME    RESOLUTION_COLUMN
--------------- --------------- --------------------
-------------------------
HR              EMPLOYEES       COMPENSATION_CG
CDRTS$COMPENSATION_CG
HR              EMPLOYEES       JOB_IDENTIFIER_CG
CDRTS$JOB_IDENTIFIER_CG
```

9

Using Procedural Replication

Learn what procedural replication is and how you can configure it.

For procedural replication concepts, see About Procedural Replication.

Topics:

- About Procedural Replication
- Procedural Replication Process Overview
  Procedural replication uses a trail record to ensure that sufficient information is encapsulated with the record.
- Enabling Procedural Replication
  Procedural replication is disabled by default. You can enable it by setting the `TRANLOGOPTIONS` option, `ENABLE_PROCEDURAL_REPLICATION`, to yes.
- Determining Whether Procedural Replication Is On
  Use the `GG_PROCEDURE_REPLICATION_ON` function in the `DBMS_GOLDENGATE_ADM` package to determine whether Oracle GoldenGate procedural replication is on or off.
- Enabling and Disabling Supplemental Logging
  Oracle GoldenGate provides GGSCI commands to allow you to enable or disable procedural supplemental logging.
- Filtering Features for Procedural Replication
  You can specify which procedures and packages you want to include or exclude for procedure replication.
- Handling Procedural Replication Errors
  Procedural replication uses `REPERROR` parameter to configure the behavior of Replicat when an procedural error occurs.
- Procedural Replication Pragma Options
  There are four pragma options for procedures: `AUTO`, `MANUAL`, `UNSUPPORTED`, and `NONE`.
- Listing the Procedures Supported for Oracle GoldenGate Procedural Replication
  The `DBA_GG_SUPPORTED_PROCEDURES` view displays information about the supported packages for Oracle GoldenGate procedural replication.
- Monitoring Oracle GoldenGate Procedural Replication
  A set of data dictionary views enable you to monitor Oracle GoldenGate procedural replication.

About Procedural Replication

Oracle GoldenGate procedural replication allows you to replicate Oracle Database supplied PL/SQL procedures avoiding the shipping and applying or high volume records usually generated by these operations. Procedural replication implements dictionary changes that control user and session behavior and the swapping of objects in dictionary.

Procedural replication is not related to the replication of the `CREATE`, `ALTER`, and `DROP` statements (or DDL), rather it is the replication of the procedure call like:

```
CALL procedure_name(arg1, arg2, ...);
```

As opposed to:

```
exec procedure_name(arg1, arg2, ...)
```

After you enable procedural replication, calls to procedures in Oracle Database supplied packages at one database are replicated to one or more other databases and then executed at those databases. For example, a call to subprograms in the `DBMS_REDEFINITION` package can perform an online redefinition of a table. If the table is replicated at several databases, and if you want the same online redefinition to be performed on the table at each database, then you can make the calls to the subprograms in the `DBMS_REDEFINITION` package at one database, and Oracle GoldenGate can replicate those calls to the other databases.

To support procedural replication, your Oracle Database should be configured to identify procedures that are enabled for this optimization.

To use procedural replication, the following prerequisites must be met:

- Oracle GoldenGate with Extract and Replicat.
- System supplied packages are only working in combination with DML and DDL.

Procedural Replication Process Overview

Procedural replication uses a trail record to ensure that sufficient information is encapsulated with the record.

To use Oracle GoldenGate procedural replication, you need to enable it. Your Oracle Database must have a built in mechanism to identify the procedures that are enabled for this optimization.

PL/SQL pragmas are used to indicate which procedures can be replicated. When the pragma is specified, a callback is made to Logminer on entry and exit from the routine. The callback provides the name of the procedure call and arguments and indicates if the procedure exited successfully or with an error. Logminer augments the redo stream with the information from the callbacks. For supported procedures, the normal redo generated by the procedure is suppressed, and only the procedure call is replicated.

A new trail record is generated to identify procedural replication. This trail record leverages existing trail column data format for arguments passed to PL/SQL procedures. For LOBs, data is passed in chunks similar to existing trail format for LOBs. This trail record has sufficient information to replay the procedure as-is on the target.

When you enable procedural replication, it prevents writing of individual records impacted by the procedure to the trail file.

If an error is encountered when applying a PL/SQL procedure, the Replicat can replay the entire PL/SQL procedure.

Enabling Procedural Replication

Procedural replication is disabled by default. You can enable it by setting the `TRANLOGOPTIONS` option, `ENABLE_PROCEDURAL_REPLICATION`, to yes.

Once you enable the procedural option for one Extract, it remains on and can not be disabled.

If you want to use Oracle GoldenGate in an Oracle Database Vault environment with procedural replication, then you must set the appropriate privileges. See *Oracle Database Vault Administrator's Guide*.

To enable procedural replication:

1. Ensure that you are in triggerless mode, see Prerequisites for Configuring DDL.
2. Connect to the source database as an Oracle GoldenGate administrator with `dblogin`.
3. Set the `TRANLOGOPTIONS` parameter option to yes.

   ```
   TRANLOGOPTIONS INTEGRATEDPARAMS (ENABLE_PROCEDURAL_REPLICATION Y)
   ```

Procedural replication is enabled for Extract.

Determining Whether Procedural Replication Is On

Use the `GG_PROCEDURE_REPLICATION_ON` function in the `DBMS_GOLDENGATE_ADM` package to determine whether Oracle GoldenGate procedural replication is on or off.

If you want to use Oracle GoldenGate in an Oracle Database Vault environment with procedural replication, then you must set the appropriate privileges. See *Oracle Database Vault Administrator's Guide*.

To enable procedural replication:

1. Connect to the database as `sys` (`sqlplus`, `sqlcl`, `sqldeveloper`) not as an Oracle GoldenGate administrator.
2. Run the `GG_PROCEDURE_REPLICATION_ON` function.

Example 9-1 Running the `GG_PROCEDURE_REPLICATION_ON` Function

```
SET SERVEROUTPUT ON
DECLARE
  on_or_off   NUMBER;
BEGIN
  on_or_off := DBMS_GOLDENGATE_ADM.GG_PROCEDURE_REPLICATION_ON;
  IF on_or_off=1 THEN
    DBMS_OUTPUT.PUT_LINE('Oracle GoldenGate procedural replication is ON.');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Oracle GoldenGate procedural replication is OFF.');
  END IF;
END;
/
```

Enabling and Disabling Supplemental Logging

Oracle GoldenGate provides GGSCI commands to allow you to enable or disable procedural supplemental logging.

To enable supplemental logging:

1. Connect to the source database as the Oracle GoldenGate administrator with dblogin.

   ```
   CONNECT https://localhost:9000 DEPLOYMENT demo AS admin PASSWORD
   adminpw
   ```

   ```
   DBLOGIN USERIDALIAS admin_dba DOMAIN OracleGoldenGate
   ```

2. Add supplemental logging for procedural replication.

   ```
   ADD PROCEDURETRANDATA
   ```

   ```
   INFO OGG-13005 PROCEDURETRANDATA supplemental logging has been
   enabled.
   ```

Supplemental logging is enabled for procedure replication.

To disable supplemental logging:

1. Connect to the source database as the Oracle GoldenGate administrator with dblogin.

   ```
   CONNECT https://localhost:9000 DEPLOYMENT demo AS admin PASSWORD
   adminpw
   ```

   ```
   DBLOGIN USERIDALIAS admin_dba DOMAIN OracleGoldenGate
   ```

2. Remove supplemental logging for procedure replication.

   ```
   DELETE PROCEDURETRANDATA
   ```

Supplemental logging is disabled for procedure replication.

To view information about supplemental logging:

1. Connect to the source database as the Oracle GoldenGate administrator with dblogin.

   ```
   CONNECT https://localhost:9000 DEPLOYMENT demo AS admin PASSWORD
   adminpw
   ```

   ```
   DBLOGIN USERIDALIAS admin_dba DOMAIN OracleGoldenGate
   ```

2. Display supplemental logging information for procedure replication.

   ```
   INFO PROCEDURETRANDATA
   ```

Supplemental logging information for procedure replication is displayed.

Filtering Features for Procedural Replication

You can specify which procedures and packages you want to include or exclude for procedure replication.

You group supported packages and procedures using feature groups. You use the procedure parameter with the `INCLUDE` or `EXCLUDE` keyword to filter features for procedure replication.

In the procedure parameter, `INCLUDE` or `EXCLUDE` specify the beginning of a filtering clause. They specify the procedures to replicate (`INCLUDE`) or filter out (`EXCLUDE`). The filtering clause must consist of the `INCLUDE ALL_SUPPORTED` or `EXCLUDE ALL_SUPPORTED` keyword followed by any valid combination of the other filtering options of the procedure parameter. The `EXCLUDE` filter takes precedence over any `INCLUDE` filters that contain the same criteria.

Note:

When replicating Oracle Streams Advanced Queuing (AQ) procedures, you must use the `RULE` option in your parameter file as follows:

```
PROCEDURE INCLUDE FEATURE ALL_SUPPORTED
```

or

```
PROCEDURE INCLUDE FEATURE AQ, RULE
```

Do *not* use `PROCEDURE INCLUDE FEATURE AQ` without the `RULE` option.

Including all system supplied packages at Extract:

1. Connect to Extract in the source database.

   ```
   EXTRACT edba
   ```

   ```
   USERIDALIAS admin_dbA DOMAIN ORADEV
   ```

2. Create a new trail file.

   ```
   EXTTRAIL ea
   ```

3. Enable procedure replication, if not already done.

   ```
   TRANLOGOPTIONS INTEGRATEDPARAMS (ENABLE_PROCEDURAL_REPLICATION Y)
   ```

4. Include filter for procedure replication.

   ```
   PROCEDURE INCLUDE FEATURE ALL_SUPPORTED
   ```

You have successfully included all system supplied packages for procedure replication.

Excluding specific packages at Replicat:

1. Connect to Replicat in the target database.

   ```
   REPLICAT rdba
   ```

   ```
   USERIDALIAS admin_dbBDOMAIN ORADEV
   ```

2. Include filter for procedure replication.

   ```
   PROCEDURE EXCLUDE FEATURE RLS
   ```

You have successfully excluded specific packages for procedure replication.

Handling Procedural Replication Errors

Procedural replication uses `REPERROR` parameter to configure the behavior of Replicat when an procedural error occurs.

By default, Replicat will abend when a procedural replication occurs so using the following steps sets up error handling:

1. Connect to Replicat in the target database.

   ```
   REPLICAT rdba
   ```

   ```
   USERIDALIAS admin_dbBDOMAIN ORADEV
   ```

2. Include filter for procedure replication.

   ```
   PROCEDURE EXCLUDE FEATURE RLS
   ```

3. Specify error handling parameter, see REPERROR in *Reference for Oracle GoldenGate* for other options.

   ```
   REPERROR (PROCEDURE, DISCARD)
   ```

You have successfully handled errors for procedural replication.

Procedural Replication Pragma Options

There are four pragma options for procedures: `AUTO`, `MANUAL`, `UNSUPPORTED`, and `NONE`.

PL/SQL enter and exit markers are logged for procedures with pragmas `AUTO`, `MANUAL`, and `UNSUPPORTED`. The redo logs generated between the enter and exit markers are grouped and discarded.

Following is a list of the packages and procedures that are pragma constructs for replication. Any package or procedure not in this list is not considered a pragma construct for PL/SQL replication and is equivalent to pragma `NONE`.

PL/SQL Procedures with Pragma are UNSUPPORTED

Procedures and packages with the pragma `UNSUPPORTED` stop apply at the point of procedure invocation so that manual intervention can be taken. The following procedures are pragma and `UNSUPPORTED`.

Schema	Package	Procedure	Pragma
SYS	DBMS_REDEFINITION	ABORT_UPDATE	PRAGMA UNSUPPORTED
SYS	DBMS_REDEFINITION	EXECUTE_UPDATE	PRAGMA UNSUPPORTED
XDB	DBMS_XDBZ	ADD_APPLICATION_PRINCIPAL	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDBZ	CHANGE_APPLICATION_MEMBERSHIP	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDBZ	DELETE_APPLICATION_PRINCIPAL	PRAGMA UNSUPPORTED with COMMIT

Schema	Package	Procedure	Pragma
XDB	DBMS_XDBZ	SET_APPLICATION_PRINCIPAL	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_ADMIN	CREATENONCEKEY	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_ADMIN	INSTALLDEFAULTWALLET	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_ADMIN	MOVEXDB_TABLESPACE	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_ADMIN	REBUILDHIERARCHICALINDEX	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_CONFIG	ADDAUTHENTICATIONMAPPING	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_CONFIG	ADDAUTHENTICATIONMETHOD	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_CONFIG	ADDTRUSTMAPPING	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_CONFIG	ADDTRUSTSCHEME	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_CONFIG	CLEARHTTPDIGESTS	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_CONFIG	DELETEAUTHENTICATIONMAPPING	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_CONFIG	DELETEAUTHENTICATIONMETHOD	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_CONFIG	DELETETRUSTMAPPING	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_CONFIG	DELETETRUSTSCHEME	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_CONFIG	ENABLECUSTOMAUTHENTICATION	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_CONFIG	ENABLECUSTOMTRUST	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_CONFIG	ENABLEDIGESTAUTHENTICATION	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_CONFIG	ISGLOBALPORTENABLED	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_CONFIG	SETDYNAMICGROUPSTORE	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_CONFIG	SETGLOBALPORTENABLED	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XDB_CONFIG	SETHTTPCONFIGREALM	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XMLINDEX	DROPPARAMETER	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XMLINDEX	MODIFYPARAMETER	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XMLINDEX	REGISTERPARAMETER	PRAGMA UNSUPPORTED with COMMIT
XDB	DBMS_XMLSCHEMA	COPYEVOLVE	PRAGMA UNSUPPORTED with COMMIT

PL/SQL Procedures with Pragma `AUTO`

For the procedures and packages with the pragma AUTO, the top-level PL/SQL API is called during apply.

Schema	Package	Procedure	Pragma
DVSYS	DBMS_MACADM	ADD_AUTH_TO_REALM	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	ADD_CMD_RULE_TO_POLICY	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	ADD_FACTOR_LINK	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	ADD_INDEX_FUNCTION	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	ADD_NLS_DATA	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	ADD_OBJECT_TO_REALM	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	ADD_OWNER_TO_POLICY	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	ADD_POLICY_FACTOR	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	ADD_REALM_TO_POLICY	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	ADD_RULE_TO_RULE_SET	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	AUTHORIZE_DATAPUMP_USER	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	AUTHORIZE_DDL	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	AUTHORIZE_DIAGNOSTIC_ADMIN	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	AUTHORIZE_MAINTENANCE_USER	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	AUTHORIZE_PREPROCESSOR	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	AUTHORIZE_PROXY_USER	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	AUTHORIZE_SCHEDULER_USER	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	AUTHORIZE_TTS_USER	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	CHANGE_IDENTITY_FACTOR	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	CHANGE_IDENTITY_VALUE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	CREATE_COMMAND_RULE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	CREATE_CONNECT_COMMAND_RULE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	CREATE_DOMAIN_IDENTITY	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	CREATE_FACTOR	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	CREATE_FACTOR_TYPE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	CREATE_IDENTITY	PRAGMA AUTO with COMMIT

Schema	Package	Procedure	Pragma
DVSYS	DBMS_MACADM	CREATE_IDENTITY_MAP	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	CREATE_MAC_POLICY	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	CREATE_POLICY	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	CREATE_POLICY_LABEL	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	CREATE_REALM	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	CREATE_ROLE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	CREATE_RULE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	CREATE_RULE_SET	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	CREATE_SESSION_EVENT_CMD_RULE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	CREATE_SESSION_EVENT_CMD_RULE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_AUTH_FROM_REALM	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_CMD_RULE_FROM_POLICY	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_COMMAND_RULE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_CONNECT_COMMAND_RULE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_FACTOR	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_FACTOR_LINK	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_FACTOR_TYPE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_IDENTITY	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_IDENTITY_MAP	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_INDEX_FUNCTION	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_MAC_POLICY_CASCADE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_OBJECT_FROM_REALM	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_OWNER_FROM_POLICY	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_POLICY_FACTOR	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_POLICY_LABEL	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_REALM	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_REALM_CASCADE	PRAGMA AUTO with COMMIT

Schema	Package	Procedure	Pragma
DVSYS	DBMS_MACADM	DELETE_REALM_FROM_POLICY	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_ROLE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_RULE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_RULE_FROM_RULE_SET	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_RULE_SET	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_SESSION_EVENT_CMD_RULE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DELETE_SYSTEM_EVENT_CMD_RULE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DISABLE_DV	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DISABLE_DV_DICTIONARY_ACCTS	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DISABLE_DV_PATCH_ADMIN_AUDIT	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DISABLE_ORADEBUG	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DROP_DOMAIN_IDENTITY	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DROP_POLICY	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	ENABLE_DV	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	ENABLE_DV_DICTIONARY_ACCTS	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	ENABLE_DV_PATCH_ADMIN_AUDIT	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	ENABLE_ORADEBUG	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	RENAME_FACTOR	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	RENAME_FACTOR_TYPE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	RENAME_POLICY	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	RENAME_REALM	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	RENAME_ROLE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	RENAME_RULE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	RENAME_RULE_SET	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UNAUTHORIZE_DATAPUMP_USER	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UNAUTHORIZE_DDL	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UNAUTHORIZE_DIAGNOSTIC_ADMIN	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UNAUTHORIZE_MAINTENANCE_USER	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UNAUTHORIZE_PREPROCESSOR	PRAGMA AUTO with COMMIT

Schema	Package	Procedure	Pragma
DVSYS	DBMS_MACADM	UNAUTHORIZE_PROXY_USER	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UNAUTHORIZE_SCHEDULER_USER	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UNAUTHORIZE_TTS_USER	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UPDATE_COMMAND_RULE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UPDATE_CONNECT_COMMAND_RULE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UPDATE_FACTOR	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UPDATE_FACTOR_TYPE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UPDATE_IDENTITY	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UPDATE_MAC_POLICY	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UPDATE_POLICY_DESCRIPTION	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UPDATE_POLICY_STATE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UPDATE_REALM	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UPDATE_REALM_AUTH	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UPDATE_ROLE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UPDATE_RULE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UPDATE_RULE_SET	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UPDATE_SESSION_EVENT_CMD_RULE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	UPDATE_SYSTEM_EVENT_CMD_RULE	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	CREATE_ADMIN_AUDIT	PRAGMA AUTO
DVSYS	DBMS_MACADM	CREATE_MACOLS_CONTEXTS	PRAGMA AUTO with COMMIT
DVSYS	DBMS_MACADM	DROP_MACOLS_CONTEXTS	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_EVENTS	AFTER_CREATE	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_EVENTS	AFTER_DROP	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_EVENTS	BEFORE_ALTER	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_LGSTNDBY_UTIL	ADD_COMPARTMENTS	PRAGMA AUTO
LBACSYS	LBAC_LGSTNDBY_UTIL	ADD_GROUPS	PRAGMA AUTO

Schema	Package	Procedure	Pragma
LBACSYS	LBAC_LGSTNDBY_UTIL	ALTER_COMPARTMENTS	PRAGMA AUTO
LBACSYS	LBAC_LGSTNDBY_UTIL	ALTER_GROUPS	PRAGMA AUTO
LBACSYS	LBAC_LGSTNDBY_UTIL	CONFIGURE_OLS	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_LGSTNDBY_UTIL	CREATE_POLICY	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_LGSTNDBY_UTIL	DISABLE_OLS	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_LGSTNDBY_UTIL	DROP_ALL_COMPARTMENTS	PRAGMA AUTO
LBACSYS	LBAC_LGSTNDBY_UTIL	DROP_ALL_GROUPS	PRAGMA AUTO
LBACSYS	LBAC_LGSTNDBY_UTIL	DROP_COMPARTMENTS	PRAGMA AUTO
LBACSYS	LBAC_LGSTNDBY_UTIL	DROP_GROUPS	PRAGMA AUTO
LBACSYS	LBAC_LGSTNDBY_UTIL	ENABLE_OLS	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_LGSTNDBY_UTIL	INSERT_LABEL	PRAGMA AUTO
LBACSYS	LBAC_LGSTNDBY_UTIL	SAVE_DEFAULT_LABELS	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_LGSTNDBY_UTIL	SET_COMPARTMENTS	PRAGMA AUTO
LBACSYS	LBAC_LGSTNDBY_UTIL	SET_DEFAULT_LABEL	PRAGMA AUTO
LBACSYS	LBAC_LGSTNDBY_UTIL	SET_GROUPS	PRAGMA AUTO
LBACSYS	LBAC_LGSTNDBY_UTIL	SET_LEVELS	PRAGMA AUTO
LBACSYS	LBAC_LGSTNDBY_UTIL	SET_ROW_LABEL	PRAGMA AUTO
LBACSYS	LBAC_LGSTNDBY_UTIL	SET_USER_LABELS	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_LGSTNDBY_UTIL	STORE_LABEL_LIST	PRAGMA AUTO
LBACSYS	LBAC_POLICY_ADMIN	ALTER_SCHEMA_POLICY	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_POLICY_ADMIN	APPLY_SCHEMA_POLICY	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_POLICY_ADMIN	APPLY_TABLE_POLICY	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_POLICY_ADMIN	DISABLE_SCHEMA_POLICY	PRAGMA AUTO with COMMIT

Schema	Package	Procedure	Pragma
LBACSYS	LBAC_POLICY_ADMIN	DISABLE_TABLE_POLICY	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_POLICY_ADMIN	ENABLE_SCHEMA_POLICY	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_POLICY_ADMIN	ENABLE_TABLE_POLICY	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_POLICY_ADMIN	POLICY_SUBSCRIBE	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_POLICY_ADMIN	POLICY_UNSUBSCRIBE	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_POLICY_ADMIN	REMOVE_SCHEMA_POLICY	PRAGMA AUTO with COMMIT
LBACSYS	LBAC_POLICY_ADMIN	REMOVE_TABLE_POLICY	PRAGMA AUTO with COMMIT
LBACSYS	SA_AUDIT_ADMIN	AUDIT	PRAGMA AUTO with COMMIT
LBACSYS	SA_AUDIT_ADMIN	AUDIT_LABEL	PRAGMA AUTO with COMMIT
LBACSYS	SA_AUDIT_ADMIN	AUDIT_LABEL_ENABLED	PRAGMA AUTO with COMMIT
LBACSYS	SA_AUDIT_ADMIN	AUDIT_LABEL_ENABLED_SQL	PRAGMA AUTO with COMMIT
LBACSYS	SA_AUDIT_ADMIN	CREATE_VIEW	PRAGMA AUTO with COMMIT
LBACSYS	SA_AUDIT_ADMIN	DROP_VIEW	PRAGMA AUTO with COMMIT
LBACSYS	SA_AUDIT_ADMIN	NOAUDIT	PRAGMA AUTO with COMMIT
LBACSYS	SA_AUDIT_ADMIN	NOAUDIT_LABEL	PRAGMA AUTO with COMMIT
LBACSYS	SA_COMPONENTS	ALTER_COMPARTMENT	PRAGMA AUTO with COMMIT
LBACSYS	SA_COMPONENTS	ALTER_COMPARTMENT	PRAGMA AUTO with COMMIT
LBACSYS	SA_COMPONENTS	ALTER_GROUP	PRAGMA AUTO with COMMIT
LBACSYS	SA_COMPONENTS	ALTER_GROUP	PRAGMA AUTO with COMMIT
LBACSYS	SA_COMPONENTS	ALTER_GROUP_PARENT	PRAGMA AUTO
LBACSYS	SA_COMPONENTS	ALTER_GROUP_PARENT	PRAGMA AUTO
LBACSYS	SA_COMPONENTS	ALTER_GROUP_PARENT	PRAGMA AUTO
LBACSYS	SA_COMPONENTS	ALTER_LEVEL	PRAGMA AUTO with COMMIT

Schema	Package	Procedure	Pragma
LBACSYS	SA_COMPONENTS	ALTER_LEVEL	PRAGMA AUTO with COMMIT
LBACSYS	SA_COMPONENTS	CREATE_COMPARTMENT	PRAGMA AUTO with COMMIT
LBACSYS	SA_COMPONENTS	CREATE_GROUP	PRAGMA AUTO
LBACSYS	SA_COMPONENTS	CREATE_LEVEL	PRAGMA AUTO with COMMIT
LBACSYS	SA_COMPONENTS	DROP_COMPARTMENT	PRAGMA AUTO with COMMIT
LBACSYS	SA_COMPONENTS	DROP_COMPARTMENT	PRAGMA AUTO with COMMIT
LBACSYS	SA_COMPONENTS	DROP_GROUP	PRAGMA AUTO with COMMIT
LBACSYS	SA_COMPONENTS	DROP_GROUP	PRAGMA AUTO with COMMIT
LBACSYS	SA_COMPONENTS	DROP_LEVEL	PRAGMA AUTO with COMMIT
LBACSYS	SA_COMPONENTS	DROP_LEVEL	PRAGMA AUTO with COMMIT
LBACSYS	SA_COMPONENTS	ALTER_LABEL	PRAGMA AUTO with COMMIT
LBACSYS	SA_COMPONENTS	ALTER_LABEL	PRAGMA AUTO with COMMIT
LBACSYS	SA_COMPONENTS	CREATE_LABEL	PRAGMA AUTO with COMMIT
LBACSYS	SA_COMPONENTS	DROP_LABEL	PRAGMA AUTO with COMMIT
LBACSYS	SA_COMPONENTS	DROP_LABEL	PRAGMA AUTO with COMMIT
LBACSYS	SA_SYSDBA	ALTER_POLICY	PRAGMA AUTO with COMMIT
LBACSYS	SA_SYSDBA	DISABLE_POLICY	PRAGMA AUTO with COMMIT
LBACSYS	SA_SYSDBA	DROP_POLICY	PRAGMA AUTO with COMMIT
LBACSYS	SA_SYSDBA	ENABLE_POLICY	PRAGMA AUTO with COMMIT
LBACSYS	SA_USER_ADMIN	DROP_USER_ACCESS	PRAGMA AUTO with COMMIT
LBACSYS	SA_USER_ADMIN	SET_PROG_PRIVS	PRAGMA AUTO with COMMIT
LBACSYS	SA_USER_ADMIN	SET_USER_PRIVS	PRAGMA AUTO with COMMIT
SYS	DBMS_AQ	AQ$_BACKGROUND_OPER	PRAGMA AUTO

Schema	Package	Procedure	Pragma
SYS	DBMS_AQ	AQ$_DELETE_DIOT_TAB	PRAGMA AUTO
SYS	DBMS_AQ	AQ$_DELETE_HIST_TAB	PRAGMA AUTO
SYS	DBMS_AQ	AQ$_DELETE_TIOT_TAB	PRAGMA AUTO
SYS	DBMS_AQ	AQ$_INSERT_DIOT_TAB	PRAGMA AUTO
SYS	DBMS_AQ	AQ$_INSERT_HIST_TAB	PRAGMA AUTO
SYS	DBMS_AQ	AQ$_INSERT_TIOT_TAB	PRAGMA AUTO
SYS	DBMS_AQ	AQ$_UPDATE_HIST_TAB	PRAGMA AUTO
SYS	DBMS_AQ	AQ$_UPDATE_HIST_TAB_EX	PRAGMA AUTO
SYS	DBMS_AQ	DEQUEUE_INTERNAL	PRAGMA AUTO
SYS	DBMS_AQ	ENQUEUE_INT_SHARD	PRAGMA AUTO
SYS	DBMS_AQ	ENQUEUE_INT_SHARD	PRAGMA AUTO
SYS	DBMS_AQ	ENQUEUE_INT_SHARD	PRAGMA AUTO
SYS	DBMS_AQ	ENQUEUE_INT_SHARD_JMS	PRAGMA AUTO
SYS	DBMS_AQ	ENQUEUE_INT_UNSHARDED	PRAGMA AUTO
SYS	DBMS_AQ	ENQUEUE_INT_UNSHARDED	PRAGMA AUTO
SYS	DBMS_AQ	ENQUEUE_INT_UNSHARDED	PRAGMA AUTO
SYS	DBMS_AQ	ENQUEUE_INT_UNSHARDED	PRAGMA AUTO
SYS	DBMS_AQ	REGISTRATION_REPLICATION	PRAGMA AUTO
SYS	DBMS_AQADM	ALTER_AQ_AGENT	PRAGMA AUTO
SYS	DBMS_AQADM	CREATE_AQ_AGENT	PRAGMA AUTO
SYS	DBMS_AQADM	DISABLE_DB_ACCESS	PRAGMA AUTO
SYS	DBMS_AQADM	DROP_AQ_AGENT	PRAGMA AUTO
SYS	DBMS_AQADM	ENABLE_DB_ACCESS	PRAGMA AUTO
SYS	DBMS_AQADM	GRANT_SYSTEM_PRIVILEGE	PRAGMA AUTO
SYS	DBMS_AQADM	GRANT_TYPE_ACCESS	PRAGMA AUTO
SYS	DBMS_AQADM	REVOKE_SYSTEM_PRIVILEGE	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	ALTER_QUEUE	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	ALTER_QUEUE_TABLE	PRAGMA AUTO

Schema	Package	Procedure	Pragma
SYS	DBMS_AQADM_SYS	ALTER_SHARDED_QUEUE	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	ALTER_SUBSCRIBER_11G	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	CREATE_EVICTION_TABLE	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	CREATE_EXCEPTION_QUEUE	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	CREATE_NP_QUEUE_INT	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	CREATE_QUEUE	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	CREATE_QUEUE_TABLE	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	CREATE_SHARDED_QUEUE	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	DROP_EVICTION_TABLE	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	DROP_QUEUE	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	DROP_QUEUE_TABLE	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	DROP_SHARDED_QUEUE_INT	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	ENABLE_JMS_TYPES_INT	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	GRANT_QUEUE_PRIVILEGE	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	MIGRATE_QUEUE_TABLE	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	PATCH_QUEUE_TABLE	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	PATCH_QUEUE_TABLE	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	PSTUPD_CREATE_EVICTION_TABLE	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	PURGE_QUEUE_TABLE_INT	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	REMOVE_ORPHMSGS_INT	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	REMOVE_SUBSCRIBER_11G_INT	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	REVOKE_QUEUE_PRIVILEGE	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	START_QUEUE	PRAGMA AUTO
SYS	DBMS_AQADM_SYS	STOP_QUEUE	PRAGMA AUTO
SYS	DBMS_AQELM	SET_MAILHOST	PRAGMA AUTO
SYS	DBMS_AQELM	SET_MAILPORT	PRAGMA AUTO
SYS	DBMS_AQELM	SET_PROXY	PRAGMA AUTO

Schema	Package	Procedure	Pragma
SYS	DBMS_AQELM	SET_SENDFROM	PRAGMA AUTO
SYS	DBMS_AQ_SYS_IMP_INTERNAL	BUMP_TID_SEQUENCE	PRAGMA AUTO
SYS	DBMS_AQ_SYS_IMP_INTERNAL	CLEANUP_SCHEMA_IMPORT	PRAGMA AUTO
SYS	DBMS_AQ_SYS_IMP_INTERNAL	IMPORT_CMT_TIME_TABLE	PRAGMA AUTO with COMMIT
SYS	DBMS_AQ_SYS_IMP_INTERNAL	IMPORT_DEQUEUELOG_TABLE	PRAGMA AUTO with COMMIT
SYS	DBMS_AQ_SYS_IMP_INTERNAL	IMPORT_EXP_ENTRY	PRAGMA AUTO
SYS	DBMS_AQ_SYS_IMP_INTERNAL	IMPORT_HISTORY_TABLE	PRAGMA AUTO with COMMIT
SYS	DBMS_AQ_SYS_IMP_INTERNAL	IMPORT_INDEX_TABLE	PRAGMA AUTO with COMMIT
SYS	DBMS_AQ_SYS_IMP_INTERNAL	IMPORT_QTAB_EXPDEP	PRAGMA AUTO
SYS	DBMS_AQ_SYS_IMP_INTERNAL	IMPORT_QUEUE	PRAGMA AUTO with COMMIT
SYS	DBMS_AQ_SYS_IMP_INTERNAL	IMPORT_QUEUE_META	PRAGMA AUTO
SYS	DBMS_AQ_SYS_IMP_INTERNAL	IMPORT_QUEUE_SEQ	PRAGMA AUTO
SYS	DBMS_AQ_SYS_IMP_INTERNAL	IMPORT_QUEUE_TABLE	PRAGMA AUTO with COMMIT
SYS	DBMS_AQ_SYS_IMP_INTERNAL	IMPORT_SIGNATURE_TABLE	PRAGMA AUTO with COMMIT
SYS	DBMS_AQ_SYS_IMP_INTERNAL	IMPORT_SUBSCRIBER_TABLE	PRAGMA AUTO with COMMIT
SYS	DBMS_AQ_SYS_IMP_INTERNAL	IMPORT_TIMEMGR_TABLE	PRAGMA AUTO with COMMIT
SYS	DBMS_AQ_SYS_IMP_INTERNAL	POST_TTS_REBUILD_IDX	PRAGMA AUTO with COMMIT
SYS	DBMS_AQ_SYS_IMP_INTERNAL	POST_TTS_SHARDED_Q	PRAGMA AUTO
SYS	DBMS_AQ_SYS_IMP_INTERNAL	POST_TTS_WORK	PRAGMA AUTO
SYS	DBMS_AQ_SYS_IMP_INTERNAL	POST_TTS_WORK_REMAINING	PRAGMA AUTO
SYS	DBMS_DBFS_CONTENT_ADMIN	EXIM_MOUNT	PRAGMA AUTO
SYS	DBMS_DBFS_CONTENT_ADMIN	EXIM_MOUNTP	PRAGMA AUTO
SYS	DBMS_DBFS_CONTENT_ADMIN	EXIM_STORE	PRAGMA AUTO

Schema	Package	Procedure	Pragma
SYS	DBMS_DBFS_CONTENT_ADMIN	MOUNTSTORE_LOG	PRAGMA AUTO
SYS	DBMS_DBFS_CONTENT_ADMIN	REGISTERSTORE_LOG	PRAGMA AUTO
SYS	DBMS_DBFS_CONTENT_ADMIN	UNMOUNTSTORE_LOG	PRAGMA AUTO
SYS	DBMS_DBFS_CONTENT_ADMIN	UNREGISTERSTORE_LOG	PRAGMA AUTO
SYS	DBMS_DBFS_SFS	NORMALIZEFS	PRAGMA AUTO with COMMIT
SYS	DBMS_DBFS_CONTENT_ADMIN	REORGANIZEFS	PRAGMA AUTO with COMMIT
SYS	DBMS_DBFS_CONTENT_ADMIN	SHRINKFS	PRAGMA AUTO with COMMIT
SYS	DBMS_DBFS_SFS_ADMIN	CREATEFILESYSTEM_LOG	PRAGMA AUTO
SYS	DBMS_DBFS_SFS_ADMIN	DELETE_ORPHANS_LOG	PRAGMA AUTO with COMMIT
SYS	DBMS_DBFS_SFS_ADMIN	DROPFILESYSTEM_LOG	PRAGMA AUTO
SYS	DBMS_DBFS_SFS_ADMIN	EXIM_ATTRV	PRAGMA AUTO with COMMIT
SYS	DBMS_DBFS_SFS_ADMIN	EXIM_FS	PRAGMA AUTO
SYS	DBMS_DBFS_SFS_ADMIN	EXIM_GRANTS	PRAGMA AUTO with COMMIT
SYS	DBMS_DBFS_SFS_ADMIN	EXIM_SEQ	PRAGMA AUTO
SYS	DBMS_DBFS_SFS_ADMIN	EXIM_SNAP	PRAGMA AUTO
SYS	DBMS_DBFS_SFS_ADMIN	EXIM_TABP	PRAGMA AUTO
SYS	DBMS_DBFS_SFS_ADMIN	EXIM_TAB_LOG	PRAGMA AUTO
SYS	DBMS_DBFS_SFS_ADMIN	EXIM_VOL	PRAGMA AUTO
SYS	DBMS_DBFS_SFS_ADMIN	INITFILESYSTEM_LOG	PRAGMA AUTO
SYS	DBMS_DBFS_SFS_ADMIN	PARTITION_SEQUENCE_LOG	PRAGMA AUTO with COMMIT
SYS	DBMS_DBFS_SFS_ADMIN	RECACHE_SEQUENCE_LOG	PRAGMA AUTO with COMMIT
SYS	DBMS_DBFS_SFS_ADMIN	REGISTERFILESYSTEM_LOG	PRAGMA AUTO
SYS	DBMS_DBFS_SFS_ADMIN	SETFSPROPERTIES_LOG	PRAGMA AUTO

Schema	Package	Procedure	Pragma
SYS	DBMS_DBFS_SFS_ADMIN	UNREGISTERFILESYSTEM_LOG	PRAGMA AUTO
SYS	DBMS_DDL	SET_TRIGGER_FIRING_PROPERTY	PRAGMA AUTO with COMMIT
SYS	DBMS_DDL	SET_TRIGGER_FIRING_PROPERTY	PRAGMA AUTO with COMMIT
SYS	DBMS_FGA	ADD_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_FGA	DISABLE_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_FGA	DROP_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_FGA	ENABLE_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_GOLDENGATE_ADM_INT_I	ADD_AUTO_CDR_COLGROUP_INT	PRAGMA AUTO with COMMIT
SYS	DBMS_FGA	ADD_AUTO_CDR_DELTA_RES_INT	PRAGMA AUTO with COMMIT
SYS	DBMS_FGA	ADD_AUTO_CDR_INT	PRAGMA AUTO with COMMIT
SYS	DBMS_FGA	ALTER_AUTO_CDR_COLGROUP_INT	PRAGMA AUTO with COMMIT
SYS	DBMS_FGA	ALTER_AUTO_CDR_INT	PRAGMA AUTO with COMMIT
SYS	DBMS_FGA	REMOVE_AUTO_CDR_COLGROUP_INT	PRAGMA AUTO with COMMIT
SYS	DBMS_FGA	REMOVE_AUTO_CDR_DELTA_RES_INT	PRAGMA AUTO with COMMIT
SYS	DBMS_FGA	REMOVE_AUTO_CDR_INT	PRAGMA AUTO with COMMIT
SYS	DBMS_GOLDENGATE_IMP	ACDR_COLUMN	PRAGMA AUTO with COMMIT
SYS	DBMS_GOLDENGATE_IMP	ACDR_COLUMN_GROUP	PRAGMA AUTO with COMMIT
SYS	DBMS_GOLDENGATE_IMP	ACDR_END	PRAGMA AUTO with COMMIT
SYS	DBMS_GOLDENGATE_IMP	ACDR_START	PRAGMA AUTO with COMMIT
SYS	DBMS_GOLDENGATE_IMP	ACDR_TABLE	PRAGMA AUTO with COMMIT
SYS	DBMS_INTERNAL_LOGSTDBY	EDS_EVOLVE_DISABLE	PRAGMA AUTO with COMMIT
SYS	DBMS_INTERNAL_LOGSTDBY	EDS_EVOLVE_ENABLE	PRAGMA AUTO with COMMIT
SYS	DBMS_INTERNAL_ROLLING	DESTROY_META	PRAGMA AUTO
SYS	DBMS_INTERNAL_ROLLING	INSERT_DGLRDDIR	PRAGMA AUTO
SYS	DBMS_INTERNAL_ROLLING	INSERT_DGLRDEVT	PRAGMA AUTO

Schema	Package	Procedure	Pragma
SYS	DBMS_INTERNAL_ROLLING	SET_UPGRADE_FLAGS	PRAGMA AUTO
SYS	DBMS_INTERNAL_ROLLING	UPDATE_DGLRDINS_PROGRESS	PRAGMA AUTO
SYS	DBMS_INTERNAL_ROLLING	UPSERT_DGLRDCON	PRAGMA AUTO
SYS	DBMS_INTERNAL_ROLLING	UPSERT_DGLRDDAT	PRAGMA AUTO
SYS	DBMS_INTERNAL_ROLLING	UPSERT_DGLRDINS	PRAGMA AUTO
SYS	DBMS_INTERNAL_ROLLING	UPSERT_DGLRDPAR	PRAGMA AUTO
SYS	DBMS_INTERNAL_ROLLING	UPSERT_DGLRDSTA	PRAGMA AUTO
SYS	DBMS_INTERNAL_ROLLING	UPSERT_DGLRDSTS	PRAGMA AUTO
SYS	DBMS_ISCHED	CREATE_CREDENTIAL	PRAGMA AUTO with COMMIT
SYS	DBMS_ISCHED	EXEC_JOB_RUN_LSA	PRAGMA AUTO
SYS	DBMS_ISCHED	SET_AGENT_REGISTRATION_PASS	PRAGMA AUTO with COMMIT
SYS	DBMS_PRVTAQIS	SUBID_REPLICATE	PRAGMA AUTO with COMMIT
SYS	DBMS_PRVTAQIS	ADD_DURABLE_SUB	PRAGMA AUTO with COMMIT
SYS	DBMS_PRVTAQIS	ALTER_SUBSCRIBER_12G	PRAGMA AUTO
SYS	DBMS_PRVTAQIS	REMOVE_SUBSCRIBER_12G	PRAGMA AUTO
SYS	DBMS_REDACT	ADD_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_REDACT	ALTER_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_REDACT	APPLY_POLICY_EXPR_TO_COL	PRAGMA AUTO with COMMIT
SYS	DBMS_REDACT	CREATE_POLICY_EXPRESSION	PRAGMA AUTO with COMMIT
SYS	DBMS_REDACT	DISABLE_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_REDACT	DROP_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_REDACT	DROP_POLICY_EXPRESSION	PRAGMA AUTO with COMMIT
SYS	DBMS_REDACT	ENABLE_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_REDACT	FPM_MASK	PRAGMA AUTO with COMMIT
SYS	DBMS_REDACT	FPM_UNMASK	PRAGMA AUTO with COMMIT
SYS	DBMS_REDACT	UPDATE_FULL_REDACTION_VALUES	PRAGMA AUTO with COMMIT
SYS	DBMS_REDACT	UPDATE_POLICY_EXPRESSION	PRAGMA AUTO with COMMIT

Schema	Package	Procedure	Pragma
SYS	DBMS_REDEFINITION	ABORT_REDEF_TABLE	PRAGMA AUTO with COMMIT
SYS	DBMS_REDEFINITION	ABORT_ROLLBACK	PRAGMA AUTO with COMMIT
SYS	DBMS_REDEFINITION	COPY_TABLE_DEPENDENTS	PRAGMA AUTO with COMMIT
SYS	DBMS_REDEFINITION	FINISH_REDEF_TABLE	PRAGMA AUTO with COMMIT
SYS	DBMS_REDEFINITION	REGISTER_DEPENDENT_OBJECT	PRAGMA AUTO with COMMIT
SYS	DBMS_REDEFINITION	ROLLBACK	PRAGMA AUTO with COMMIT
SYS	DBMS_REDEFINITION	SET_PARAM	PRAGMA AUTO with COMMIT
SYS	DBMS_REDEFINITION	START_REDEF_TABLE	PRAGMA AUTO with COMMIT
SYS	DBMS_REDEFINITION	SYNC_INTERIM_TABLE	PRAGMA AUTO with COMMIT
SYS	DBMS_REDEFINITION	UNREGISTER_DEPENDENT_OBJECT	PRAGMA AUTO with COMMIT
SYS	DBMS_RLS_INT	ADD_GROUPED_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_RLS_INT	ADD_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_RLS_INT	ADD_POLICY_CONTEXT	PRAGMA AUTO with COMMIT
SYS	DBMS_RLS_INT	ALTER_GROUPED_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_RLS_INT	ALTER_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_RLS_INT	CREATE_POLICY_GROUP	PRAGMA AUTO with COMMIT
SYS	DBMS_RLS_INT	DELETE_POLICY_GROUP	PRAGMA AUTO with COMMIT
SYS	DBMS_RLS_INT	DISABLE_GROUPED_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_RLS_INT	DROP_GROUPED_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_RLS_INT	DROP_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_RLS_INT	DROP_POLICY_CONTEXT	PRAGMA AUTO with COMMIT
SYS	DBMS_RLS_INT	ENABLE_GROUPED_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_RLS_INT	ENABLE_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_RLS_INT	REFRESH_GROUPED_POLICY	PRAGMA AUTO with COMMIT
SYS	DBMS_RLS_INT	REFRESH_POLICY	PRAGMA AUTO with COMMIT

Schema	Package	Procedure	Pragma
SYS	DBMS_RULEADM_INTERNAL	ADD_RULE	PRAGMA AUTO
SYS	DBMS_RULEADM_INTERNAL	ALTER_EVALUATION_CONTEXT	PRAGMA AUTO
SYS	DBMS_RULEADM_INTERNAL	ALTER_RULE	PRAGMA AUTO
SYS	DBMS_RULEADM_INTERNAL	CREATE_EVALUATION_CONTEXT	PRAGMA AUTO
SYS	DBMS_RULEADM_INTERNAL	CREATE_RULE	PRAGMA AUTO
SYS	DBMS_RULEADM_INTERNAL	CREATE_RULE_SET	PRAGMA AUTO
SYS	DBMS_RULEADM_INTERNAL	DROP_EVALUATION_CONTEXT	PRAGMA AUTO
SYS	DBMS_RULEADM_INTERNAL	DROP_RULE	PRAGMA AUTO
SYS	DBMS_RULEADM_INTERNAL	DROP_RULE_SET	PRAGMA AUTO
SYS	DBMS_RULEADM_INTERNAL	REMOVE_RULE	PRAGMA AUTO
SYS	DBMS_RULE_ADM	GRANT_OBJECT_PRIVILEGE	PRAGMA AUTO
SYS	DBMS_RULE_ADM	GRANT_SYSTEM_PRIVILEGE	PRAGMA AUTO
SYS	DBMS_RULE_ADM	REVOKE_OBJECT_PRIVILEGE	PRAGMA AUTO
SYS	DBMS_RULE_ADM	REVOKE_SYSTEM_PRIVILEGE	PRAGMA AUTO
SYS	DBMS_SCHEDULER	ADD_EVENT_QUEUE_SUBSCRIBER	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	ADD_GROUP_MEMBER	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	ADD_JOB_EMAIL_NOTIFICATION	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	ADD_TO_INCOMPATIBILITY	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	ADD_WINDOW_GROUP_MEMBER	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	ALTER_CHAIN	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	ALTER_CHAIN	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	ALTER_RUNNING_CHAIN	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	ALTER_RUNNING_CHAIN	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	ANALYZE_CHAIN	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	AUTO_PURGE	PRAGMA AUTO with COMMIT

Schema	Package	Procedure	Pragma
SYS	DBMS_SCHEDULER	CHECK_CREDENTIAL	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	COPY_JOB	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_CHAIN	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_DATABASE_DESTINATION	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_EVENT_SCHEDULE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_FILE_WATCHER	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_GROUP	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_INCOMPATIBILITY	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_JOB	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_JOB	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_JOB	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_JOB	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_JOB	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_JOB	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_JOBS	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_JOBS	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_JOB_CLASS	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_PROGRAM	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_RESOURCE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_SCHEDULE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_WINDOW	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_WINDOW	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	CREATE_WINDOW_GROUP	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DEFINE_ANYDATA_ARGUMENT	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DEFINE_CHAIN_EVENT_STEP	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DEFINE_CHAIN_EVENT_STEP	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DEFINE_CHAIN_RULE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DEFINE_CHAIN_STEP	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DEFINE_METADATA_ARGUMENT	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DEFINE_PROGRAM_ARGUMENT	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DEFINE_PROGRAM_ARGUMENT	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DELETE_FILE	PRAGMA AUTO with COMMIT

Schema	Package	Procedure	Pragma
SYS	DBMS_SCHEDULER	DISABLE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DISABLE1_CALENDAR_CHECK	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DROP_AGENT_DESTINATION	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DROP_CHAIN	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DROP_CHAIN_RULE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DROP_CHAIN_STEP	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DROP_CREDENTIAL	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DROP_DATABASE_DESTINATION	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DROP_FILE_WATCHER	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DROP_GROUP	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DROP_INCOMPATIBILITY	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DROP_JOB	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DROP_JOB_CLASS	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DROP_PROGRAM	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DROP_PROGRAM_ARGUMENT	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DROP_PROGRAM_ARGUMENT	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DROP_RESOURCE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DROP_SCHEDULE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DROP_WINDOW	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	DROP_WINDOW_GROUP	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	ENABLE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	END_DETACHED_JOB_RUN	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	EVALUATE_RUNNING_CHAIN	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	GET_AGENT_INFO	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	GET_ATTRIBUTE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	GET_ATTRIBUTE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	GET_ATTRIBUTE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	GET_ATTRIBUTE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	GET_ATTRIBUTE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	GET_ATTRIBUTE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	GET_ATTRIBUTE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	GET_ATTRIBUTE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	GET_ATTRIBUTE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	GET_FILE	PRAGMA AUTO with COMMIT

Schema	Package	Procedure	Pragma
SYS	DBMS_SCHEDULER	GET_FILE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	GET_FILE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	GET_SCHEDULER_ATTRIBUTE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	PURGE_LOG	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	PUT_FILE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	PUT_FILE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	PUT_FILE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	REMOVE_EVENT_QUEUE_SUBSCRIBER	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	REMOVE_FROM_INCOMPATIBILITY	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	REMOVE_GROUP_MEMBER	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	REMOVE_JOB_EMAIL_NOTIFICATION	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	REMOVE_WINDOW_GROUP_MEMBER	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	RESET_JOB_ARGUMENT_VALUE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	RESET_JOB_ARGUMENT_VALUE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	RUN_CHAIN	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	RUN_CHAIN	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	SET_ATTRIBUTE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	SET_ATTRIBUTE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	SET_ATTRIBUTE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	SET_ATTRIBUTE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	SET_ATTRIBUTE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	SET_ATTRIBUTE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	SET_ATTRIBUTE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	SET_ATTRIBUTE_NULL	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	SET_JOB_ANYDATA_VALUE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	SET_JOB_ANYDATA_VALUE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	SET_JOB_ARGUMENT_VALUE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	SET_JOB_ARGUMENT_VALUE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	SET_JOB_ATTRIBUTES	PRAGMA AUTO with COMMIT

Schema	Package	Procedure	Pragma
SYS	DBMS_SCHEDULER	SET_RESOURCE_CONSTRAINT	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	SET_SCHEDULER_ATTRIBUTE	PRAGMA AUTO with COMMIT
SYS	DBMS_SCHEDULER	SHOW_ERRORS	PRAGMA AUTO with COMMIT
SYS	DBMS_SQL_TRANSLATOR	CLEAR_SQL_TRANSLATION_ERROR	PRAGMA AUTO with COMMIT
SYS	DBMS_SQL_TRANSLATOR	CREATE_PROFILE	PRAGMA AUTO with COMMIT
SYS	DBMS_SQL_TRANSLATOR	DEREGISTER_ERROR_TRANSLATION	PRAGMA AUTO with COMMIT
SYS	DBMS_SQL_TRANSLATOR	DEREGISTER_SQL_TRANSLATION	PRAGMA AUTO with COMMIT
SYS	DBMS_SQL_TRANSLATOR	DROP_PROFILE	PRAGMA AUTO with COMMIT
SYS	DBMS_SQL_TRANSLATOR	ENABLE_ERROR_TRANSLATION	PRAGMA AUTO with COMMIT
SYS	DBMS_SQL_TRANSLATOR	ENABLE_SQL_TRANSLATION	PRAGMA AUTO with COMMIT
SYS	DBMS_SQL_TRANSLATOR	REGISTER_ERROR_TRANSLATION	PRAGMA AUTO with COMMIT
SYS	DBMS_SQL_TRANSLATOR	REGISTER_SQL_TRANSLATION	PRAGMA AUTO with COMMIT
SYS	DBMS_SQL_TRANSLATOR	SET_ATTRIBUTE	PRAGMA AUTO with COMMIT
SYS	DBMS_SQL_TRANSLATOR	SET_ERROR_TRANSLATION_COMMENT	PRAGMA AUTO with COMMIT
SYS	DBMS_SQL_TRANSLATOR	SET_SQL_TRANSLATION_COMMENT	PRAGMA AUTO with COMMIT
SYS	DBMS_SQL_TRANSLATOR	SET_SQL_TRANSLATION_MODULE	PRAGMA AUTO with COMMIT
SYS	DBMS_XDS	ALTER_STATIC_ACL_REFRESH	PRAGMA AUTO
SYS	DBMS_XDS	DISABLE_OLAP_POLICY	PRAGMA AUTO
SYS	DBMS_XDS	DISABLE_XDS	PRAGMA AUTO
SYS	DBMS_XDS	DROP_OLAP_POLICY	PRAGMA AUTO
SYS	DBMS_XDS	DROP_XDS	PRAGMA AUTO
SYS	DBMS_XDS	ENABLE_OLAP_POLICY	PRAGMA AUTO
SYS	DBMS_XDS	ENABLE_XDS	PRAGMA AUTO
SYS	DBMS_XDS	PURGE_ACL_REFRESH_HISTORY	PRAGMA AUTO
SYS	DBMS_XDS	SCHEDULE_STATIC_ACL_REFRESH	PRAGMA AUTO

Schema	Package	Procedure	Pragma
SYS	DBMS_XDS	SET_TRACE_LEVEL	PRAGMA AUTO
SYS	DBMS_XDS	XDS$REFRESH_STATIC_ACL	PRAGMA AUTO
SYS	LOGSTDBY_INTERNAL	EDS_EVOLVE_TABLE_I	PRAGMA AUTO with COMMIT
SYS	LOGSTDBY_INTERNAL	EDS_REMOVE_TABLE_I	PRAGMA AUTO with COMMIT
SYS	XS_ACL	ADD_ACL_PARAMETER	PRAGMA AUTO
SYS	XS_ACL	ADD_ACL_PARAMETER	PRAGMA AUTO
SYS	XS_ACL	APPEND_ACES	PRAGMA AUTO
SYS	XS_ACL	APPEND_ACES	PRAGMA AUTO
SYS	XS_ACL	CREATE_ACL	PRAGMA AUTO
SYS	XS_ACL	DELETE_ACL	PRAGMA AUTO
SYS	XS_ACL	REMOVE_ACES	PRAGMA AUTO
SYS	XS_ACL	REMOVE_ACL_PARAMETERS	PRAGMA AUTO
SYS	XS_ACL	REMOVE_ACL_PARAMETERS	PRAGMA AUTO
SYS	XS_ACL	REMOVE_ACL_PARAMETERS	PRAGMA AUTO
SYS	XS_ACL	SET_DESCRIPTION	PRAGMA AUTO
SYS	XS_ACL	SET_PARENT_ACL	PRAGMA AUTO
SYS	XS_ACL	SET_SECURITY_CLASS	PRAGMA AUTO
SYS	XS_ADMIN_UTIL	GRANT_SYSTEM_PRIVILEGE	PRAGMA AUTO
SYS	XS_ADMIN_UTIL	REVOKE_SYSTEM_PRIVILEGE	PRAGMA AUTO
SYS	XS_DATA_SECURITY	ADD_COLUMN_CONSTRAINTS	PRAGMA AUTO
SYS	XS_DATA_SECURITY	ADD_COLUMN_CONSTRAINTS	PRAGMA AUTO
SYS	XS_DATA_SECURITY	APPEND_REALM_CONSTRAINTS	PRAGMA AUTO
SYS	XS_DATA_SECURITY	APPEND_REALM_CONSTRAINTS	PRAGMA AUTO
SYS	XS_DATA_SECURITY	APPLY_OBJECT_POLICY	PRAGMA AUTO
SYS	XS_DATA_SECURITY	CREATE_ACL_PARAMETER	PRAGMA AUTO
SYS	XS_DATA_SECURITY	CREATE_POLICY	PRAGMA AUTO
SYS	XS_DATA_SECURITY	DELETE_ACL_PARAMETER	PRAGMA AUTO

Schema	Package	Procedure	Pragma
SYS	XS_DATA_SECURITY	DELETE_POLICY	PRAGMA AUTO
SYS	XS_DATA_SECURITY	DISABLE_OBJECT_POLICY	PRAGMA AUTO
SYS	XS_DATA_SECURITY	ENABLE_OBJECT_POLICY	PRAGMA AUTO
SYS	XS_DATA_SECURITY	REMOVE_COLUMN_CONSTRAINTS	PRAGMA AUTO
SYS	XS_DATA_SECURITY	REMOVE_OBJECT_POLICY	PRAGMA AUTO
SYS	XS_DATA_SECURITY	REMOVE_REALM_CONSTRAINTS	PRAGMA AUTO
SYS	XS_DATA_SECURITY	SET_DESCRIPTION	PRAGMA AUTO
SYS	XS_NAMESPACE	ADD_ATTRIBUTES	PRAGMA AUTO
SYS	XS_DATA_SECURITY	ADD_ATTRIBUTES	PRAGMA AUTO
SYS	XS_DATA_SECURITY	CREATE_TEMPLATE	PRAGMA AUTO
SYS	XS_DATA_SECURITY	DELETE_TEMPLATE	PRAGMA AUTO
SYS	XS_DATA_SECURITY	REMOVE_ATTRIBUTES	PRAGMA AUTO
SYS	XS_DATA_SECURITY	REMOVE_ATTRIBUTES	PRAGMA AUTO
SYS	XS_DATA_SECURITY	REMOVE_ATTRIBUTES	PRAGMA AUTO
SYS	XS_DATA_SECURITY	SET_DESCRIPTION	PRAGMA AUTO
SYS	XS_DATA_SECURITY	SET_HANDLER	PRAGMA AUTO
SYS	XS_PRINCIPAL	ADD_PROXY_TO_DBUSER	PRAGMA AUTO
SYS	XS_PRINCIPAL	ADD_PROXY_USER	PRAGMA AUTO
SYS	XS_PRINCIPAL	ADD_PROXY_USER	PRAGMA AUTO
SYS	XS_PRINCIPAL	CREATE_DYNAMIC_ROLE	PRAGMA AUTO
SYS	XS_PRINCIPAL	CREATE_ROLE	PRAGMA AUTO
SYS	XS_PRINCIPAL	CREATE_USER	PRAGMA AUTO
SYS	XS_PRINCIPAL	DELETE_PRINCIPAL	PRAGMA AUTO
SYS	XS_PRINCIPAL	ENABLE_BY_DEFAULT	PRAGMA AUTO
SYS	XS_PRINCIPAL	ENABLE_ROLES_BY_DEFAULT	PRAGMA AUTO
SYS	XS_PRINCIPAL	GRANT_ROLES	PRAGMA AUTO
SYS	XS_PRINCIPAL	GRANT_ROLES	PRAGMA AUTO

Schema	Package	Procedure	Pragma
SYS	XS_PRINCIPAL	REMOVE_PROXY_FROM_DBUSER	PRAGMA AUTO
SYS	XS_PRINCIPAL	REMOVE_PROXY_USERS	PRAGMA AUTO
SYS	XS_PRINCIPAL	REMOVE_PROXY_USERS	PRAGMA AUTO
SYS	XS_PRINCIPAL	REVOKE_ROLES	PRAGMA AUTO
SYS	XS_PRINCIPAL	REVOKE_ROLES	PRAGMA AUTO
SYS	XS_PRINCIPAL	REVOKE_ROLES	PRAGMA AUTO
SYS	XS_PRINCIPAL	SET_ACL	PRAGMA AUTO
SYS	XS_PRINCIPAL	SET_DESCRIPTION	PRAGMA AUTO
SYS	XS_PRINCIPAL	SET_DYNAMIC_ROLE_DURATION	PRAGMA AUTO
SYS	XS_PRINCIPAL	SET_DYNAMIC_ROLE_SCOPE	PRAGMA AUTO
SYS	XS_PRINCIPAL	SET_EFFECTIVE_DATES	PRAGMA AUTO
SYS	XS_PRINCIPAL	SET_GUID	PRAGMA AUTO
SYS	XS_PRINCIPAL	SET_PROFILE	PRAGMA AUTO
SYS	XS_PRINCIPAL	SET_USER_SCHEMA	PRAGMA AUTO
SYS	XS_PRINCIPAL	SET_USER_STATUS	PRAGMA AUTO
SYS	XS_PRINCIPAL_INT	SET_VERIFIER_HELPER	PRAGMA AUTO
SYS	XS_ROLESET	ADD_ROLES	PRAGMA AUTO
SYS	XS_ROLESET	ADD_ROLES	PRAGMA AUTO
SYS	XS_ROLESET	CREATE_ROLESET	PRAGMA AUTO
SYS	XS_ROLESET	DELETE_ROLESET	PRAGMA AUTO
SYS	XS_ROLESET	REMOVE_ROLES	PRAGMA AUTO
SYS	XS_ROLESET	REMOVE_ROLES	PRAGMA AUTO
SYS	XS_ROLESET	REMOVE_ROLES	PRAGMA AUTO
SYS	XS_ROLESET	SET_DESCRIPTION	PRAGMA AUTO
SYS	XS_SECURITY_CLASS	ADD_IMPLIED_PRIVILEGES	PRAGMA AUTO
SYS	XS_SECURITY_CLASS	ADD_IMPLIED_PRIVILEGES	PRAGMA AUTO
SYS	XS_SECURITY_CLASS	ADD_PARENTS	PRAGMA AUTO
SYS	XS_SECURITY_CLASS	ADD_PARENTS	PRAGMA AUTO
SYS	XS_SECURITY_CLASS	ADD_PRIVILEGES	PRAGMA AUTO
SYS	XS_SECURITY_CLASS	ADD_PRIVILEGES	PRAGMA AUTO

Schema	Package	Procedure	Pragma
SYS	XS_SECURITY_CLASS	CREATE_SECURITY_CLASS	PRAGMA AUTO
SYS	XS_SECURITY_CLASS	DELETE_SECURITY_CLASS	PRAGMA AUTO
SYS	XS_SECURITY_CLASS	REMOVE_IMPLIED_PRIVILEGES	PRAGMA AUTO
SYS	XS_SECURITY_CLASS	REMOVE_IMPLIED_PRIVILEGES	PRAGMA AUTO
SYS	XS_SECURITY_CLASS	REMOVE_IMPLIED_PRIVILEGES	PRAGMA AUTO
SYS	XS_SECURITY_CLASS	REMOVE_PARENTS	PRAGMA AUTO
SYS	XS_SECURITY_CLASS	REMOVE_PARENTS	PRAGMA AUTO
SYS	XS_SECURITY_CLASS	REMOVE_PARENTS	PRAGMA AUTO
SYS	XS_SECURITY_CLASS	REMOVE_PRIVILEGES	PRAGMA AUTO
SYS	XS_SECURITY_CLASS	REMOVE_PRIVILEGES	PRAGMA AUTO
SYS	XS_SECURITY_CLASS	REMOVE_PRIVILEGES	PRAGMA AUTO
SYS	XS_SECURITY_CLASS	SET_DESCRIPTION	PRAGMA AUTO
SYS	DBMS_RESCONFIG	ADDREPOSITORYRESCONFIG	PRAGMA AUTO with COMMIT
SYS	DBMS_RESCONFIG	ADDRESCONFIG	PRAGMA AUTO
SYS	DBMS_RESCONFIG	APPENDRESCONFIG	PRAGMA AUTO
SYS	DBMS_RESCONFIG	DELETEREPOSITORYRESCONFIG	PRAGMA AUTO with COMMIT
SYS	DBMS_RESCONFIG	DELETERESCONFIG	PRAGMA AUTO
SYS	DBMS_RESCONFIG	DELETERESCONFIG	PRAGMA AUTO
SYS	DBMS_XDBZ	DISABLE_HIERARCHY	PRAGMA AUTO with COMMIT
SYS	DBMS_XDBZ	ENABLE_HIERARCHY	PRAGMA AUTO with COMMIT
SYS	DBMS_XDB_VERSION	CHECKIN_INT	PRAGMA AUTO
SYS	DBMS_XDB_VERSION	CHECKOUT	PRAGMA AUTO
SYS	DBMS_XDB_VERSION	MAKEVERSIONED_INT	PRAGMA AUTO
SYS	DBMS_XDB_VERSION	UNCHECKOUT_INT	PRAGMA AUTO
SYS	DBMS_XLSB	DELETERESOURCE	PRAGMA AUTO
SYS	DBMS_XLSB	DELNAMELOCKS	PRAGMA AUTO
SYS	DBMS_XLSB	INSERTRESOURCE	PRAGMA AUTO

Schema	Package	Procedure	Pragma
SYS	DBMS_XLSB	INSERTRESOURCENXOB	PRAGMA AUTO
SYS	DBMS_XLSB	INSERTRESOURCENXOBCLOB	PRAGMA AUTO
SYS	DBMS_XLSB	INSERTRESOURCEREF	PRAGMA AUTO
SYS	DBMS_XLSB	INSERTTOHTABLE	PRAGMA AUTO
SYS	DBMS_XLSB	INSERTTOUSERHTAB	PRAGMA AUTO
SYS	DBMS_XLSB	LINKRESOURCE	PRAGMA AUTO
SYS	DBMS_XLSB	SAVEACL	PRAGMA AUTO
SYS	DBMS_XLSB	SETREFCOUNT	PRAGMA AUTO
SYS	DBMS_XLSB	TOUCHOID	PRAGMA AUTO

PL/SQL Procedures with Pragma MANUAL

For the procedures and packages pragma-ed MANUAL, the top-level PL/SQL API is not called.

Schema	Package	Procedure	Pragma
SYS	DBMS_AQ	AQ$_BACKGROUND_OPER_PAS	PRAGMA MANUAL
SYS	DBMS_AQ	DEQUEUE_INTERNAL_PAS	PRAGMA MANUAL
SYS	DBMS_AQ	ENQUEUE_INT_UNSHARDED_PAS	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	ALTER_PROPAGATION_SCHEDULE_INT	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	ALTER_QUEUE_INT	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	ALTER_QUEUE_TABLE_INT	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	ALTER_SUBSCRIBER_11G_INT	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	CREATE_QUEUE_INT	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	CREATE_QUEUE_TABLE_INT	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	DISABLE_PROP_SCHEDULE_INT	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	DROP_QUEUE_INT	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	DROP_QUEUE_TABLE_INT	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	ENABLE_PROP_SCHEDULE_INT	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	GRANT_QUEUE_PRIVILEGE_INT	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	MIGRATE_QUEUE_TABLE_INT	PRAGMA MANUAL

Schema	Package	Procedure	Pragma
SYS	DBMS_AQADM_SYS	PURGE_QUEUE_TABLE	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	RECOVER_PROPAGATION_INT	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	REMOVE_ORPHMSGS_NR	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	REMOVE_SUBSCRIBER_11G	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	REVOKE_QUEUE_PRIVILEGE_INT	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	SCHEDULE_PROPAGATION_INT	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	START_QUEUE_INT	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	STOP_QUEUE_INT	PRAGMA MANUAL
SYS	DBMS_AQADM_SYS	UNSCHEDULE_PROPAGATION_INT	PRAGMA MANUAL
SYS	DBMS_GOLDENGATE_AUTH	GRANT_ADMIN_PRIVILEGE	PRAGMA MANUAL with COMMIT
SYS	DBMS_GOLDENGATE_AUTH	REVOKE_ADMIN_PRIVILEGE	PRAGMA MANUAL with COMMIT
SYS	DBMS_INTERNAL_LOGSTDBY	EDS_EVOLVE_TABLE_START	PRAGMA MANUAL with COMMIT
SYS	DBMS_PRVTAQIS	SUBID_REPLICATE_INT	PRAGMA MANUAL
SYS	LOGSTDBY_INTERNAL	EDS_ADD_TABLE_I	PRAGMA MANUAL with COMMIT
SYS	XS_ADMIN_UTIL	DROP_SCHEMA_OBJECTS	PRAGMA MANUAL
XDB	DBMS_XDBZ0	DISABLE_HIERARCHY_INTERNAL	PRAGMA MANUAL
XDB	DBMS_XDBZ0	ENABLE_HIERARCHY_INTERNAL	PRAGMA MANUAL

PL/SQL Procedures with Pragma `NONE`

For the procedures and packages pragma-ed `NONE`, PL/SQL markers are not generated and no grouping is performed. Redo logs generated by these procedures are applied or skipped based on table level replication semantics.

Schema	Package	Procedure	Pragma
DVSYS	DBMS_MACADM	DISABLE_EVENT	PRAGMA NONE
DVSYS	DBMS_MACADM	DV_SANITY_CHECK	PRAGMA NONE
DVSYS	DBMS_MACADM	ENABLE_EVENT	PRAGMA NONE
DVSYS	DBMS_MACADM	SET_PRESERVE_CASE	PRAGMA NONE
DVSYS	DBMS_MACADM	INIT_SESSION	PRAGMA NONE

Schema	Package	Procedure	Pragma
DVSYS	DBMS_MACADM	UPDATE_POLICY_LABEL_CONTEXT	PRAGMA NONE
DVSYS	DBMS_MACOLS_SESSION	LABEL_AUDIT_RAISE	PRAGMA NONE
DVSYS	DBMS_MACOLS_SESSION	RESTORE_DEFAULT_LABELS	PRAGMA NONE
DVSYS	DBMS_MACOLS_SESSION	SET_POLICY_LABEL_CONTEXT	PRAGMA NONE
DVSYS	DBMS_MACOUT	DISABLE	PRAGMA NONE
DVSYS	DBMS_MACOUT	ENABLE	PRAGMA NONE
DVSYS	DBMS_MACOUT	PL	PRAGMA NONE
DVSYS	DBMS_MACOUT	PUT_LINE	PRAGMA NONE
DVSYS	DBMS_MACOUT	SET_FACTOR	PRAGMA NONE
DVSYS	DBMS_MACSEC_ROLES	SET_ROLE	PRAGMA NONE
DVSYS	DBMS_MACSEC_ROLES	EVALUATE	PRAGMA NONE
DVSYS	DBMS_MACSEC_ROLES	EVALUATE_TR	PRAGMA NONE
DVSYS	DBMS_MACSEC_ROLES	EVALUATE_WR	PRAGMA NONE
DVSYS	DBMS_MACUTL	CHECK_DVSYS_DML_ALLOWED	PRAGMA NONE
DVSYS	DBMS_MACUTL	RAISE_ERROR	PRAGMA NONE
DVSYS	DBMS_MACUTL	RAISE_UNAUTHORIZED_OPERATION	PRAGMA NONE
DVSYS	EVENT	SET	PRAGMA NONE
DVSYS	EVENT	SETDEFAULT	PRAGMA NONE
DVSYS	EVENT	SET_C	PRAGMA NONE
SYS	DBMS_AQ	AQ$_DEQUEUE	PRAGMA NONE
SYS	DBMS_AQ	AQ$_DEQUEUE	PRAGMA NONE
SYS	DBMS_AQ	AQ$_DEQUEUE	PRAGMA NONE
SYS	DBMS_AQ	AQ$_DEQUEUE	PRAGMA NONE
SYS	DBMS_AQ	BIND_AGENT	PRAGMA NONE
SYS	DBMS_AQ	DEQUEUE	PRAGMA NONE
SYS	DBMS_AQ	DEQUEUE	PRAGMA NONE
SYS	DBMS_AQ	DEQUEUE	PRAGMA NONE
SYS	DBMS_AQ	ENQUEUE	PRAGMA NONE
SYS	DBMS_AQ	ENQUEUE	PRAGMA NONE
SYS	DBMS_AQ	ENQUEUE	PRAGMA NONE
SYS	DBMS_AQ	LISTEN	PRAGMA NONE
SYS	DBMS_AQ	LISTEN	PRAGMA NONE
SYS	DBMS_AQ	POST	PRAGMA NONE

Schema	Package	Procedure	Pragma
SYS	DBMS_AQ	REGISTER	PRAGMA NONE
SYS	DBMS_AQ	UNBIND_AGENT	PRAGMA NONE
SYS	DBMS_AQ	UNREGISTER	PRAGMA NONE
SYS	DBMS_AQADM	ADD_ALIAS_TO_LDAP	PRAGMA NONE
SYS	DBMS_AQADM	ADD_CONNECTION_TO_LDAP	PRAGMA NONE
SYS	DBMS_AQADM	ADD_CONNECTION_TO_LDAP	PRAGMA NONE
SYS	DBMS_AQADM	ADD_SUBSCRIBER	PRAGMA NONE
SYS	DBMS_AQADM	ALTER_PROPAGATION_SCHEDULE	PRAGMA NONE
SYS	DBMS_AQADM	ALTER_QUEUE	PRAGMA NONE
SYS	DBMS_AQADM	ALTER_QUEUE_TABLE	PRAGMA NONE
SYS	DBMS_AQADM	ALTER_SHARDED_QUEUE	PRAGMA NONE
SYS	DBMS_AQADM	ALTER_SUBSCRIBER	PRAGMA NONE
SYS	DBMS_AQADM	ALTER_SUBSCRIBER	PRAGMA NONE
SYS	DBMS_AQADM	CREATE_EXCEPTION_QUEUE	PRAGMA NONE
SYS	DBMS_AQADM	CREATE_NP_QUEUE	PRAGMA NONE
SYS	DBMS_AQADM	CREATE_QUEUE	PRAGMA NONE
SYS	DBMS_AQADM	CREATE_QUEUE_TABLE	PRAGMA NONE
SYS	DBMS_AQADM	CREATE_SHARDED_QUEUE	PRAGMA NONE
SYS	DBMS_AQADM	DEL_ALIAS_FROM_LDAP	PRAGMA NONE
SYS	DBMS_AQADM	DEL_CONNECTION_FROM_LDAP	PRAGMA NONE
SYS	DBMS_AQADM	DISABLE_PROPAGATION_SCHEDULE	PRAGMA NONE
SYS	DBMS_AQADM	DROP_QUEUE	PRAGMA NONE
SYS	DBMS_AQADM	DROP_QUEUE_TABLE	PRAGMA NONE
SYS	DBMS_AQADM	DROP_SHARDED_QUEUE	PRAGMA NONE
SYS	DBMS_AQADM	ENABLE_JMS_TYPES	PRAGMA NONE
SYS	DBMS_AQADM	ENABLE_PROPAGATION_SCHEDULE	PRAGMA NONE
SYS	DBMS_AQADM	GET_PROP_SEQNO	PRAGMA NONE
SYS	DBMS_AQADM	GET_REPLAY_INFO	PRAGMA NONE
SYS	DBMS_AQADM	GET_TYPE_INFO	PRAGMA NONE
SYS	DBMS_AQADM	GET_TYPE_INFO	PRAGMA NONE
SYS	DBMS_AQADM	GET_WATERMARK	PRAGMA NONE

Schema	Package	Procedure	Pragma
SYS	DBMS_AQADM	GRANT_QUEUE_PRIVILEGE	PRAGMA NONE
SYS	DBMS_AQADM	MIGRATE_QUEUE_TABLE	PRAGMA NONE
SYS	DBMS_AQADM	NONREPUDIATE_RECEIVER	PRAGMA NONE
SYS	DBMS_AQADM	NONREPUDIATE_RECEIVER	PRAGMA NONE
SYS	DBMS_AQADM	NONREPUDIATE_SENDER	PRAGMA NONE
SYS	DBMS_AQADM	NONREPUDIATE_SENDER	PRAGMA NONE
SYS	DBMS_AQADM	PURGE_QUEUE_TABLE	PRAGMA NONE
SYS	DBMS_AQADM	RECOVER_PROPAGATION	PRAGMA NONE
SYS	DBMS_AQADM	REMOVE_SUBSCRIBER	PRAGMA NONE
SYS	DBMS_AQADM	RESET_REPLAY_INFO	PRAGMA NONE
SYS	DBMS_AQADM	REVOKE_QUEUE_PRIVILEGE	PRAGMA NONE
SYS	DBMS_AQADM	SCHEDULE_PROPAGATION	PRAGMA NONE
SYS	DBMS_AQADM	SET_WATERMARK	PRAGMA NONE
SYS	DBMS_AQADM	START_QUEUE	PRAGMA NONE
SYS	DBMS_AQADM	START_TIME_MANAGER	PRAGMA NONE
SYS	DBMS_AQADM	STOP_QUEUE	PRAGMA NONE
SYS	DBMS_AQADM	STOP_TIME_MANAGER	PRAGMA NONE
SYS	DBMS_AQADM	UNSCHEDULE_PROPAGATION	PRAGMA NONE
SYS	DBMS_AQADM	VERIFY_QUEUE_TYPES	PRAGMA NONE
SYS	DBMS_AQADM	VERIFY_QUEUE_TYPES_GET_NRP	PRAGMA NONE
SYS	DBMS_AQADM	VERIFY_QUEUE_TYPES_NO_QUEUE	PRAGMA NONE
SYS	DBMS_AQELM	GET_MAILHOST	PRAGMA NONE
SYS	DBMS_AQELM	GET_MAILPORT	PRAGMA NONE
SYS	DBMS_AQELM	GET_PROXY	PRAGMA NONE
SYS	DBMS_AQELM	GET_SENDFROM	PRAGMA NONE
SYS	DBMS_AQELM	GET_TXTIMEOUT	PRAGMA NONE
SYS	DBMS_AQELM	HTTP_SEND	PRAGMA NONE
SYS	DBMS_AQELM	SEND_EMAIL	PRAGMA NONE
SYS	DBMS_AQIN	AQ$_DEQUEUE_IN	PRAGMA NONE
SYS	DBMS_AQIN	AQ$_DEQUEUE_IN	PRAGMA NONE

Schema	Package	Procedure	Pragma
SYS	DBMS_AQIN	AQ$_DEQUEUE_IN	PRAGMA NONE
SYS	DBMS_AQIN	AQ$_DEQUEUE_IN	PRAGMA NONE
SYS	DBMS_AQIN	AQ$_DEQUEUE_IN	PRAGMA NONE
SYS	DBMS_AQIN	AQ$_DEQUEUE_RAW	PRAGMA NONE
SYS	DBMS_AQIN	AQ$_DEQUEUE_RAW	PRAGMA NONE
SYS	DBMS_AQIN	AQ$_ENQUEUE_OBJ	PRAGMA NONE
SYS	DBMS_AQIN	AQ$_ENQUEUE_OBJ	PRAGMA NONE
SYS	DBMS_AQIN	AQ$_ENQUEUE_OBJ_NO_RECPL	PRAGMA NONE
SYS	DBMS_AQIN	AQ$_ENQUEUE_OBJ_NO_RECPL	PRAGMA NONE
SYS	DBMS_AQIN	AQ$_ENQUEUE_RAW	PRAGMA NONE
SYS	DBMS_AQIN	AQ$_JMS_ENQUEUE_BYTES_MESSAGE	PRAGMA NONE
SYS	DBMS_AQIN	AQ$_JMS_ENQUEUE_MAP_MESSAGE	PRAGMA NONE
SYS	DBMS_AQIN	AQ$_JMS_ENQUEUE_OBJECT_MESSAGE	PRAGMA NONE
SYS	DBMS_AQIN	AQ$_JMS_ENQUEUE_STREAM_MESSAGE	PRAGMA NONE
SYS	DBMS_AQIN	AQ$_JMS_ENQUEUE_TEXT_MESSAGE	PRAGMA NONE
SYS	DBMS_AQIN	AQ$_LISTEN	PRAGMA NONE
SYS	DBMS_AQIN	AQ$_QUEUE_SUBSCRIBERS	PRAGMA NONE
SYS	DBMS_AQIN	SET_DEQ_SORT	PRAGMA NONE
SYS	DBMS_AQIN	SET_MULTI_RETRY	PRAGMA NONE
SYS	DBMS_AQJMS	AQ$_GET_PROP_STAT	PRAGMA NONE
SYS	DBMS_AQJMS	AQ$_GET_TRANS_TYPE	PRAGMA NONE
SYS	DBMS_AQJMS	AQ$_REGISTER	PRAGMA NONE
SYS	DBMS_AQJMS	AQ$_UNREGISTER	PRAGMA NONE
SYS	DBMS_AQJMS	AQ$_UPDATE_PROP_STAT_QNAME	PRAGMA NONE
SYS	DBMS_AQJMS	CLEAR_DBSESSION_GUID	PRAGMA NONE
SYS	DBMS_AQJMS	CLEAR_GLOBAL_AQCLNTDB_CTX_CLNT	PRAGMA NONE
SYS	DBMS_AQJMS	CLEAR_GLOBAL_AQCLNTDB_CTX_DB	PRAGMA NONE
SYS	DBMS_AQJMS	GET_DB_USERNAME_FOR_AGENT	PRAGMA NONE
SYS	DBMS_AQJMS	SET_DBSESSION_GUID	PRAGMA NONE

Schema	Package	Procedure	Pragma
SYS	DBMS_AQJMS	SET_GLOBAL_AQCLNTDB_CTX	PRAGMA NONE
SYS	DBMS_AQJMS	SUBSCRIBER_EXISTS	PRAGMA NONE
SYS	DBMS_AQJMS	SUBSCRIBER_EXISTS	PRAGMA NONE
SYS	DBMS_ISCHED	GET_AGENT_PASS_VERIFIER	PRAGMA NONE
SYS	DBMS_ISCHED	OBFUSCATE_CREDENTIAL_PASSWORD	PRAGMA NONE
SYS	DBMS_REDEFINITION	CAN_REDEF_TABLE	PRAGMA NONE
SYS	DBMS_REDEFINITION	REDEF_TABLE	PRAGMA NONE
SYS	DBMS_SCHEDULER	CHECK_SYS_PRIVS	PRAGMA NONE
SYS	DBMS_SCHEDULER	CLOSE_WINDOW	PRAGMA NONE
SYS	DBMS_SCHEDULER	CREATE_CALENDAR_STRING	PRAGMA NONE
SYS	DBMS_SCHEDULER	CREATE_CREDENTIAL	PRAGMA NONE
SYS	DBMS_SCHEDULER	EVALUATE_CALENDAR_STRING	PRAGMA NONE
SYS	DBMS_SCHEDULER	FILE_WATCH_FILTER	PRAGMA NONE
SYS	DBMS_SCHEDULER	GENERATE_EVENT_LIST	PRAGMA NONE
SYS	DBMS_SCHEDULER	GENERATE_JOB_NAME	PRAGMA NONE
SYS	DBMS_SCHEDULER	GET_AGENT_VERSION	PRAGMA NONE
SYS	DBMS_SCHEDULER	GET_CHAIN_RULE_ACTION	PRAGMA NONE
SYS	DBMS_SCHEDULER	GET_CHAIN_RULE_CONDITION	PRAGMA NONE
SYS	DBMS_SCHEDULER	GET_DEFAULT_VALUE	PRAGMA NONE
SYS	DBMS_SCHEDULER	GET_JOB_STEP_CF	PRAGMA NONE
SYS	DBMS_SCHEDULER	GET_SYS_TIME_ZONE_NAME	PRAGMA NONE
SYS	DBMS_SCHEDULER	GET_VARCHAR2_VALUE	PRAGMA NONE
SYS	DBMS_SCHEDULER	GET_VARCHAR2_VALUE	PRAGMA NONE
SYS	DBMS_SCHEDULER	IS_SCHEDULER_CREATED_AGENT	PRAGMA NONE
SYS	DBMS_SCHEDULER	OPEN_WINDOW	PRAGMA NONE
SYS	DBMS_SCHEDULER	RESOLVE_CALENDAR_STRING	PRAGMA NONE
SYS	DBMS_SCHEDULER	RESOLVE_CALENDAR_STRING	PRAGMA NONE
SYS	DBMS_SCHEDULER	RESOLVE_NAME	PRAGMA NONE

Schema	Package	Procedure	Pragma
SYS	DBMS_SCHEDULER	RUN_JOB	PRAGMA NONE
SYS	DBMS_SCHEDULER	SET_AGENT_REGISTRATION_PASS	PRAGMA NONE
SYS	DBMS_SCHEDULER	STIME	PRAGMA NONE
SYS	DBMS_SCHEDULER	STOP_JOB	PRAGMA NONE
SYS	DBMS_SCHEDULER	SUBMIT_REMOTE_EXTERNAL_JOB	PRAGMA NONE
SYS	XS_PRINCIPAL	SET_PASSWORD	PRAGMA NONE
SYS	XS_PRINCIPAL	SET_VERIFIER	PRAGMA NONE

Listing the Procedures Supported for Oracle GoldenGate Procedural Replication

The `DBA_GG_SUPPORTED_PROCEDURES` view displays information about the supported packages for Oracle GoldenGate procedural replication.

When a procedure is supported and Oracle GoldenGate procedural replication is on, calls to the procedure are replicated, unless the procedure is excluded specifically.

1. Connect to the database as `sys` (`sqlplus`, `sqlcl`, `sqldeveloper`) not as an Oracle GoldenGate administrator.
2. Query the `DBA_GG_SUPPORTED_PROCEDURES` view.

Example 9-2   Displaying Information About the Packages Supported for Oracle GoldenGate Procedural Replication

This query displays the following information about the packages:

- The owner of each package
- The name of each package
- The name of each procedure
- The minimum database release from which the procedure is supported
- Whether there is an exclusion rule that prevents the procedure from being replicated for some database objects

```
COLUMN OWNER FORMAT A10
COLUMN PACKAGE_NAME FORMAT A15
COLUMN PROCEDURE_NAME FORMAT A15
COLUMN MIN_DB_VERSION FORMAT A14
COLUMN EXCLUSION_RULE_EXISTS FORMAT A14

SELECT OWNER,
       PACKAGE_NAME,
       PROCEDURE_NAME,
       MIN_DB_VERSION,
       EXCLUSION_RULE_EXISTS
  FROM DBA_GG_SUPPORTED_PROCEDURES;
```

Your output looks similar to the following:

```
OWNER      PACKAGE_NAME    PROCEDURE_NAME  MIN_DB_VERSION EXCLUSION_RULE
---------- --------------- --------------- -------------- --------------
XDB        DBMS_XDB_CONFIG ADDTRUSTMAPPING 12.2           NO
CTXSYS     CTX_DDL         ALTER_INDEX     12.2           NO
SYS        DBMS_FGA        DROP_POLICY     12.2           NO
SYS        XS_ACL          DELETE_ACL      12.2           NO
.
.
.
```

Monitoring Oracle GoldenGate Procedural Replication

A set of data dictionary views enable you to monitor Oracle GoldenGate procedural replication.

You can use the following views to monitor Oracle GoldenGate procedural replication:

View	Description
`DBA_GG_SUPPORTED_PACKAGES`	Provides details about supported packages for Oracle GoldenGate procedural replication. When a package is supported and Oracle GoldenGate procedural replication is on, calls to subprograms in the package are replicated.
`DBA_GG_SUPPORTED_PROCEDURES`	Provides details about the procedures that are supported for Oracle GoldenGate procedural replication.
`DBA_GG_PROC_OBJECT_EXCLUSION`	Provides details about all database objects that are on the exclusion list for Oracle GoldenGate procedural replication. A database object is added to the exclusion list using the `INSERT_PROCREP_EXCLUSION_OBJ` procedure in the `DBMS_GOLDENGATE_ADM` package. When a database object is on the exclusion list, execution of a subprogram n the package is not replicated if the subprogram operates on the excluded object.

1. Connect to the database as `sys` (`sqlplus`, `sqlcl`, or `sqldeveloper`) not as an Oracle GoldenGate administrator.
2. Query the views related to Oracle GoldenGate procedural replication.

10

Using Oracle GoldenGate with Autonomous Database

You can replicate data to Oracle Autonomous Database using Oracle GoldenGate.

Topics:

About Capturing and Replicating Data Using Autonomous Databases

You can capture changes from the Oracle Autonomous Database instance and replicate to any **target database** or platform that Oracle GoldenGate supports, including another Oracle Autonomous Database instance.

See Autonomous Database Quickstart Workshop to know more.

Use Case: When Using Oracle GoldenGate with Autonomous Databases

Using Oracle GoldenGate in the Oracle Autonomous Database can be configured to support the following scenarios:

- **Scalable Active-Active architecture**: Synchronize changes made across two or more databases to scale out workloads, provide increase resilience and near instantaneous failover across multiple data centers or regions.
- **Real-Time Data Warehouse**: Provide continuous, real-time capture and delivery of changed data between Oracle Autonomous Database systems.
- **Big Data Integration**: With Oracle GoldenGate for Big Data you can replicate data from the Oracle Autonomous Database to provide real-time streaming integration to all platforms supported by Big Data targets.
- **Real-Time Streaming Analytics**: Oracle GoldenGate integrates seamlessly with Oracle Stream Analytics to enable users to identify events of interest by executing queries against event streams in real time. It allows creating custom operational dashboards that provide real-time monitoring, transform streaming data, or raise alerts based on stream analysis.
- **Hybrid Replication**: Oracle GoldenGate replicates data from the Oracle Autonomous Database instance back to on-premise or to another cloud database or platform.

The following features are not available with Always Free Autonomous Databases:

- Supplemental logging

- Oracle GoldenGate Extract

See Always Free Autonomous Database for details.

Details of Support When Using Oracle GoldenGate with Autonomous Databases

Review the supported data types and limitations before replicating data to an Oracle Autonomous Database.

Oracle GoldenGate Replicat Limitations for Autonomous Databases

These are the limitations of Oracle GoldenGate when replicating to or from the Oracle Autonomous Database.

Supported Replicats
The following combinations of Replicats are supported in different modes when using Oracle GoldenGate with Oracle Autonomous Database:

- Parallel Replicat in integrated mode is supported for Oracle Autonomous Database Serverless.
- Classic and coordinated Replicats in integrated mode are not supported for Oracle Autonomous Database.
- Classic, coordinated, and parallel Replicats in non-integrated mode are supported for Oracle Autonomous Database.

Data Type Limitations for DDL and DML Replication
See the section Non-Supported Oracle Data Types.
Also see Limitation on the Use of Certain Data Types in the *Autonomous Database on Dedicated Exadata Infrastructure Documentation* and Data Types in the *Using Oracle Autonomous Database Serverless* guide.
DDL replication is supported depending on the restrictions in the Autonomous Databases.

Details of Support for Archived Log Retention
The two types of Autonomous Databases, Oracle Autonomous Database Serverless and Oracle Autonomous Database on Dedicated Exadata Infrastructure have different log retention behavior.

- Oracle Autonomous Database Serverless: Archived log files are kept in Fast Recovery Area (FRA) for up to 48 hours. After that, it is purged and the archived log files are moved to NFS mount storage, which is accessible by logminer. Three copies are created. The logminer should be able to access any of the copies. This is transparent to Oracle GoldenGate Extract. After it reaches 7 days, the NFS mounted copy is permanently removed. The Extract abends with the `archived log unavailable` error if the required archived log file is older than 7 days.
- Oracle Autonomous Database on Dedicated Exadata Infrastructure: When Active Data Guard (ADG) or Oracle GoldenGate is enabled, archived log files are kept in Fast Recovery Area (FRA) for up to 7 days. After that, the files are purged. There is no NFS mount location available for logminer to access archived log files that

are older than 7 days. The Extract abends with the `archived log unavailable` error if the required archived log file is older than 7 days.

> **Note:**
>
> If the database instance is closed for more than 15 minutes, then the retention time is set back to 3 days. This implies that retention of archived log files is confirmed only for 3 days, regardless of whether the database instance is closed. The files are retained for 7 days only if the database instance is not closed.

Configuring Extract to Capture from an Autonomous Database

Oracle Autonomous Database has a tight integration with Oracle GoldenGate. There are a number of differences when setting up Extract for an Autonomous database instance compared to a traditional Oracle Database.

Oracle Autonomous Database security has been enhanced to ensure that Extract is only able to capture changes from the specific tenant it connected to. However, downstream Extract is not supported.

Before You Begin

Before you start the process of capturing data from the Autonomous Database using Oracle GoldenGate you must first:

1. Unlock the pre-created Oracle GoldenGate database user `ggadmin` in the Autonomous Database.
2. Obtain the Autonomous Database client credentials to connect to the database instance.

Topics:

- Establishing Oracle GoldenGate Credentials
- Prerequisites for Configuring Oracle GoldenGate Extract to Capture from Autonomous Databases
- Configure Extract to Capture from an Autonomous Database

Establishing Oracle GoldenGate Credentials

To capture from an Autonomous Database only the `GGADMIN` account is used. The `GGADMIN` account is created inside the database when the Autonomous Database is provisioned. This account is locked. It must be unlocked before it can be used with Oracle GoldenGate. This account is the same account used for both Extracts and Replicats in the Autonomous Database.

Run the `ALTER USER` command to unlock the `ggadmin` user and set the password for it. See Creating Users with Autonomous Database with Client-Side Tools.

This `ALTER USER` command must be run by the `admin` account user for Autonomous Databases.

```
ALTER USER ggadmin IDENTIFIED BY PASSWORD ACCOUNT UNLOCK;
```

Prerequisites for Configuring Oracle GoldenGate Extract to Capture from Autonomous Databases

Prior to configuring and starting the Extract process to capture from the Autonomous Database, make sure that the following requirements are met:

- Oracle Autonomous Database environment is provisioned and running.
- Autonomous Database-level supplemental logging should be enabled by the `ADMIN` or `GGADMIN`.

Configuring Autonomous Database Supplemental Logging for Extract

To add minimal supplemental logging to your Autonomous Database instance, log into the instance as `GGADMIN` or `ADMIN` account and execute the following commands:

```
ALTER PLUGGABLE DATABASE ADD SUPPLEMENTAL LOG DATA;
```

To `DROP` Autonomous Database-level supplemental logging incase you decide to stop capturing from that database instance:

```
ALTER PLUGGABLE DATABASE DROP SUPPLEMENTAL
        LOG DATA;
```

You can verify that the Autonomous Database-level supplemental logging is configured properly by issuing this SQL statement:

```
SELECT MINIMAL FROM dba_supplemental_logging;
```

The output for this statement is:

```
MINIMAL
-------
YES
```

The `MINIMAL` column will be `YES` if supplemental logging has been correctly set for this Autonomous Database instance.

Configure Extract to Capture from an Autonomous Database

Following are the steps to configure an Extract to capture from an Oracle Autonomous Database :

1. Install Oracle GoldenGate for your Oracle Autonomous Database instance.
2. Obtain Autonomous Database Client Credentials.

   To establish connection to your Oracle Autonomous Database instance, download the client credentials file. To download client credentials, you can use the Oracle Cloud Infrastructure Console or Database Actions Launchpad. See Downloading Client Credentials (Wallets).

> **Note:**
>
> If you do not have administrator access to the Autonomous Database you should ask your service administrator to download and provide the credentials files to you.

The following steps use the **Database Actions Launchpad** to download the client credentials.

a. Log in to your Oracle Autonomous Database account.

b. From the **Database Instance** page, click **Database Actions**. This launches the Database Actions Launchpad. The Launchpad attempts to log you into the database as ADMIN. If that is not successful, you will be prompted for your database ADMIN username and password.

c. On the **Database Actions** Launchpad, under **Administration**, click **Download Client Credentials (Wallets)**

d. Enter a password to secure your Client Credentials zip file and click **Download**.

> **Note:**
>
> The password you provide when you download the wallet protects the downloaded Client Credentials wallet.

e. Save the credentials `zip` file to your local system.

The credentials `zip` file contains the following files:

- `cwallet.sso`
- `ewallet.p12`
- `keystore.jks`
- `ojdbc.properties`
- `sqlnet.ora`
- `tnsnames.ora`
- `truststore.jks`
- `ewallet.pem`
- `README.txt`

Refer and update (if required) the `sqlnet.ora` and `tnsnames.ora` files while configuring Oracle GoldenGate to work with the Autonomous Database instance.

3. Configure the server where Oracle GoldenGate is running to connect to the Autonomous Database instance.

   a. Log in to the server where Oracle GoldenGate was installed.

   b. Transfer the credentials `zip` file that you downloaded from Oracle Autonomous database instance to the Oracle GoldenGate server.

c. In the Oracle GoldenGate server, unzip the credentials file into a new directory, for example: `/u02/data/adwc_credentials`. This is your key directory.

d. To configure the connection details, open your `tnsnames.ora` file from the Oracle client location in the Oracle GoldenGate instance.

e. Use the connection string with the `LOW` consumer group *`dbname`*`_low`, for example, `graphdb1_low`, and move it to your local `tnsnames.ora` file.

   See *Local Naming Parameters in the tnsnames.ora File* chapter in the *Oracle Database Net Services Reference* guide.

   Note:

   The `tnsnames.ora` file provided with the credentials file contains three database service names identifiable as:

   ```
   ADWC_Database_Name_low
   ADWC_Database_Name_medium
   ADWC_Database_Name_high
   ```

   Oracle recommends that you use *`ADWC_Database_Name`*`_low` with Oracle GoldenGate. See Predefined Database Service Names for Autonomous Database in the *Using Oracle Autonomous Database Serverless* guide or Predefined Database Service Names for Autonomous Databases for Oracle Autonomous Database on Dedicated Exadata Infrastructure.

f. Edit the `tnsnames.ora` file in the Oracle GoldenGate instance to include the connection details available in the `tnsnames.ora` file in your key directory (the directory where you unzipped the credentials `zip` file downloaded from the Autonomous Database.

   ```
   Sample Connection String
   adw1_low. = (description=
                    (retry_count=20)(retry_delay=3)
                    (address=(protocol=tcps)(port=1522)(host=adb-
   preprod.us-phoenix-1.oraclecloud.com))

   (connect_data=(service_name=okd2ybgcz4mjx94_graphdb1_low.adb.orac
   lecloud.com))
                    (security=(ssl_server_cert_dn="CN=adwc-
   preprod.uscom-east-1.oraclecloud.com,OU=Oracle BMCS US,O=Oracle
   Corporation,L=Redwood City,ST=California,C=US"))
                 )
   ```

   If the database is within a firewall protected environment, you might not have direct access to the database. With an existing HTTP Proxy, you can pass the firewall with the following modifications to the `sqlnet.ora` and `tnsnames.ora`:

   - *`sqlnet parameters`*
   - *`address modification of tns_alias`*

If Extract becomes unresponsive due to a network timeout or connection loss, then you can add the following into the connection profile in the `tnsnames.ora` file:

```
(DESCRIPTION = (RECV_TIMEOUT=30) (ADDRESS_LIST =
      (LOAD_BALANCE=off)(FAILOVER=on)(CONNECT_TIMEOUT=3)
(RETRY_COUNT=3) (ADDRESS = (PROTOCOL = TCP)(HOST = adb-preprod.us-
phoenix-1.oraclecloud.com)(PORT = 1522))
```

g. To configure the wallet, create a `sqlnet.ora` file in the Oracle client location in the Oracle GoldenGate instance.

```
cd /u02/data/oci/network/admin
ls
sqlnet.ora tnsnames.ora
```

See Autonomous Database Client Credentials in *Using Oracle GoldenGate on Oracle Cloud Marketplace*.

h. Edit this `sqlnet.ora` file to include your key directory.

```
WALLET_LOCATION = (SOURCE = (METHOD = file) (METHOD_DATA =
(DIRECTORY="/u02/data/adwc_credentials")))
SSL_SERVER_DN_MATCH=yes
```

4. Start GGSCI.
`./ggsci`

5. Create credentials for the Extract database (or a user with same privileges). In this case, `GGADMIN` is the user and will be used to connect to the Autonomous Database, and perform commands that require a database connection. It will also be used in the `USERIDALIAS` parameter for the Extract database connection.

```
ALTER CREDENTIALSTORE ADD USER
ggadmin@dbgraph1_low PASSWORD complex_password alias adb_alias
```

6. Connect to the database using `DBLOGIN`. The `DBLOGIN` user should be the `adb_alias` account user.

```
DBLOGIN USERIDALIAS adb_alias
```

7. Configure supplemental logging on the tables, which you want to capture using `ADD TRANDATA` or `ADD SCHEMATRANDATA`. Remember that you are connected directly to the database instance, so there is no need to include the database name in these commands. Here's an exmaple:

```
ADD TRANDATA HR.EMP
```

or

```
ADD SCHEMATRANDATA HR
```

See Prerequisites for Configuring Oracle GoldenGate Extract to Capture from Autonomous Databases.

8. Add heartbeat table.

   ```
   ADD HEARTBEATTABLE
   ```

9. Add and configure an Extract to capture from the Oracle Autonomous Database instance. See Configuring Primary Extract for steps to create an Extract.

   Oracle GoldenGate Extract is designed to work with the Oracle Autonomous Database instance to ensure that it only captures from a specific database instance. This means that the database instance name is not needed for any `TABLE` or `MAP` statements.

   The following example creates an Extract (required for capturing from an Oracle Autonomous Database) called `exte`, and instructs it to begin now.

   ```
   ADD EXTRACT exte, INTEGRATED TRANLOG, BEGIN NOW
   ```

   To capture specific tables, use the two part object names.. For example, to capture from the table `HR.EMP`, in your Oracle Autonomous Database instance, use this entry in the Extract parameter file.

   ```
   TABLE HR.EMP;
   ```

   If you want to replicate `HR.EMP` into `COUNTRY.EMPLOYEE`, then your map statement would look like this:

   ```
   MAP HR.EMP, TARGET COUNTRY.EMPLOYEE;
   ```

10. Register Extract with the Oracle Autonomous Database instance. For example, to register an Extract named `exte`, use the following command:

    ```
    REGISTER EXTRACT exte DATABASE
    ```

11. You can now start your Extract and perform data replication to the Oracle Autonomous Database instance. Here's an example:

    ```
    START EXTRACT exte
    ```

    This completes the process of configuring an Extract for Oracle Autonomous Database and you can use it like any other Extract process.

Configuring Replicat to Apply to Autonomous Databases

You can replicate into the Autonomous Database from any source database or platform that is supported by Oracle GoldenGate.

Topics:

- Prerequisites for Configuring Oracle GoldenGate Replicat to an Autonomous Database
  Learn about the prerequisites for configuring Oracle GoldenGate data replication to Autonmous Databases.
- Configure Replicat to Apply to an Autonomous Database

Prerequisites for Configuring Oracle GoldenGate Replicat to an Autonomous Database

Learn about the prerequisites for configuring Oracle GoldenGate data replication to Autonmous Databases.

You should have the following details available with you:

- Your source database with Oracle GoldenGate Extract processes configured and writing trails to where the Replicat is running to apply data to the Autonomous Database target.
- Oracle Autonomous Database environment provisioned and running.

To deliver data to the Autonomous database instance using Oracle GoldenGate, perform the following tasks:

- Configure Oracle GoldenGate Replicat for an Autonomous Database
  Learn the steps to configure Oracle GoldenGate Replicat for an Autonomous Databases.
- Obtain the Autonomous Database Client Credentials
  Learn how to establish connection to your Autonomous Databases.

Configure Oracle GoldenGate Replicat for an Autonomous Database

Learn the steps to configure Oracle GoldenGate Replicat for an Autonomous Databases.

Here are the steps to complete the configuration tasks:

> **Note:**
>
> Instructions are based on the assumption that the source environment is already configured. Learn the steps required to configure replication into the Autonomous Database environment.

1. For Oracle GoldenGate on-premises, make sure that Oracle GoldenGate is installed.

   Oracle GoldenGate Classic Architecture support Autonomous Database capture using Marketplace for Oracle Autonomous Database Serverless
2. Start GGSCI.

   ```
   ./ggsci
   ```
3. The Autonomous Database instance has a pre-created user created for Oracle GoldenGate on-premise called `ggadmin`. The `ggadmin` user has been granted the required privileges for Replicat to work. This is the user where any objects used for Oracle GoldenGate processing will be stored, like the checkpoint table and heartbeat objects. By default, this user is locked. To unlock the `ggadmin` user, connect to the Oracle Autonomous Database instance as the `ADMIN` user using any SQL client tool. See Create Users on Autonomous Database with Database Actions.
4. Run the `ALTER USER` command to unlock the `ggadmin` user and set the password for it. This will be used in GGSCI for any `DBLOGIN` operations on the Autonomous Database. It will be used in Replicat to allow Oracle GoldenGate to connect to the Autonomous

Database and apply data. See Create Users on Autonomous Database with Database Actions.

```
ALTER USER ggadmin IDENTIFIED BY p0$$word ACCOUNT UNLOCK;
```

Obtain the Autonomous Database Client Credentials

Learn how to establish connection to your Autonomous Databases.

To establish a connection with an Oracle Autonomous Database instance, you need to download the client credentials files. There are two ways to download the client credentials files: the Oracle Cloud Infrastructure Console or Database Actions Launchpad. See Downloading Client Credentials (Wallets).

> **Note:**
>
> If you do not have administrator access to the Oracle Autonomous Database, you should ask your service administrator to download and provide the credentials files to you.

The following steps use the **Database Actions Launchpad** to download the client credentials files.

1. Log into your Autonomous Database account.
2. From the **Database Instance** page, click **Database Actions**. This launches the Database Actions Launchpad. The Launchpad attempts to log you into the database as ADMIN. If that is not successful, you will be prompted for your database ADMIN username and password.
3. On the **Database Actions** Launchpad, under **Administration**, click **Download Client Credentials (Wallets)**.
4. Enter a password to secure your Client Credentials zip file and click **Download**.

   > **Note:**
   >
   > The password you provide when you download the wallet protects the downloaded Client Credentials wallet.

5. Save the credentials ZIP file to your local system. The credentials ZIP file contains the following files:

- `cwallet.sso`
- `ewallet.p12`
- `keystore.jks`
- `ojdbc.properties`
- `sqlnet.ora`
- `tnsnames.ora`
- `truststore.jks`

- `ewallet.pem`
- `README.txt`

Refer and update (if required) the `sqlnet.ora` and `tnsnames.ora` files while configuring Oracle GoldenGate to work with the Oracle Autonomous Database instance.

Configure Replicat to Apply to an Autonomous Database

This section assumes that the source environment is already configured and provides the steps required to establish replication in the Oracle Autonomous Database environment.

In the Oracle GoldenGate instance, you need to complete the following:

1. Follow the steps given in Prerequisites for Configuring Oracle GoldenGate Replicat to an Autonomous Database.
2. Follow the steps given in Configure Oracle GoldenGate Replicat for an Autonomous Database.
3. Follow the steps given in Obtain the Autonomous Database Client Credentials.
4. Log into the server where Oracle GoldenGate was installed.
5. Transfer the credentials `zip` file that you downloaded from Oracle Autonomous Database to your Oracle GoldenGate instance.
6. In the Oracle GoldenGate instance, unzip the credentials file into a new directory `/u02/data/adwc_credentials`. This is your key directory.
7. To configure the connection details, open your `tnsnames.ora` file from the Oracle client location in the Oracle GoldenGate instance.

   ```
   cd /u02/data/adwc_credentials
   ls
   tnsnames.ora
   ```

8. Edit the `tnsnames.ora` file in the Oracle GoldenGate instance to include the connection details available in the `tnsnames.ora` file in your key directory (the directory where you unzipped the credentials `zip` file downloaded from Oracle Autonomous Database).

   ```
   Sample Connection String
   graphdb1_low = (description=
                   (retry_count=20)(retry_delay=3)(address=(protocol=tcps)
   (port=1522)(host=adb-preprod.us-phoenix-1.oraclecloud.com))

   (connect_data=(service_name=okd2ybgcz4mjx94_graphdb1_low.adb.oraclecloud.c
   om))
                   (security=(ssl_server_cert_dn="CN=adwc-preprod.uscom-
   east-1.oraclecloud.com,OU=Oracle BMCS US,O=Oracle Corporation,L=Redwood
   City,ST=California,C=US")))
   ```

   If Replicat becomes unresponsive due to a network timeout or lost connection, then you can add the following into the connection profile in the `tnsnames.ora` file:

   ```
   (DESCRIPTION =  (RECV_TIMEOUT=120)   (ADDRESS_LIST =
         (LOAD_BALANCE=off)(FAILOVER=on)(CONNECT_TIMEOUT=3)(RETRY_COUNT=3)
   ```

```
      (ADDRESS = (PROTOCOL = TCP)(HOST = adb-preprod.us-
phoenix-1.oraclecloud.com)(PORT = 1522))
```

> **Note:**
>
> The `tnsnames.ora` file provided with the credentials file contains three database service names identifiable as:
>
> ```
> ADWC_Database_Name_low
> ADWC_Database_Name_medium
> ADWC_Database_Name_high
> ```
>
> For Oracle GoldenGate replication, use *`ADWC_Database_Name`*`_low`.

9. To configure the wallet, create a `sqlnet.ora` file in the Oracle client location in the Oracle GoldenGate instance.

   ```
   cd /u02/data/oci/network/admin
   ls
   sqlnet.ora tnsnames.ora
   ```

10. Edit this `sqlnet.ora` file to include your key directory.

    ```
    WALLET_LOCATION = (SOURCE = (METHOD = file) (METHOD_DATA =
    (DIRECTORY="/u02/data/adwc_credentials")))
    SSL_SERVER_DN_MATCH=yes
    ```

11. Use GGSCI to log into the Oracle GoldenGate deployment.
12. Create a credential to store the `GGADMIN` user and password for the Replicat to use. For example:

    ```
    ADD CREDENTIALSTORE ALTER CREDENTIALSTORE ADD USER
    ggadmin@databasename_low PASSWORD complex_password alias adb_alias
    ```

13. Add and configure a Replicat to deliver to Oracle Autonomous Database. When creating the Replicat, use the alias created in the previous step. For setting up your Replicat and other processes, see #unique_241.

    > **Note:**
    >
    > You can use classic Replicat, coordinated Replicat, and parallel Replicat in non-integrated mode. Parallel Replicat in integrated mode is also supported for Oracle Autonomous Database Serverless.

14. You can now start your Replicat and perform data replication to the Autonomous Database.

Note:

Oracle Autonomous Database times out and disconnects the Replicat when it is idle for more than 60 minutes. When Replicat tries to apply changes (when it gets new changes) after being idle, it encounters a database error and abends. Oracle recommends that you configure Oracle GoldenGate with the `AUTORESTART` profile to avoid having to manually restart a Replicat when it times out.

11 Understanding What's Supported

This chapter contains support information for Oracle GoldenGate on Oracle Database.

Topics:

Details of Support for Oracle Data Types and Objects

This topic describes data types, objects and operations that are supported by Oracle GoldenGate.

Within the database, you can use the Dictionary view `DBA_GOLDENGATE_SUPPORT_MODE` to get information about supported objects. There are different types for replication support:

- Support by Capturing from Redo
- Procedural Replication Support

Most data types are supported (`SUPPORT_MODE=FULL`), which imply that Oracle GoldenGate captures the changes out of the redo. In some unique cases, the information cannot be captured, but the information can be fetched with a connection to the database (`SUPPORT_MODE=ID KEY`).

From Oracle GoldenGate 21c onward, DML on tables that are not supported will be automatically skipped when `DBA_GOLDENGATE_SUPPORT_MODE.SUPPORT_MODE= NONE` is set. However, DDLs for these objects are still captured based on the `DDL INCLUDE`/`EXCLUDE` settings. See Supported Objects and Operations in Oracle DDL for DDL support.

Tables supported with `ID KEY` require a connection to the source database or an ADG Standby database for fetching to support those tables. If using downstream Extract, with `NOUSERID` you must specify a `FETCHUSERID` or `FETCHUSERIDALIAS` connection.

Other changes can be replicated with Procedural Replication (`SUPPORT_MODE=PLSQL`) that requires additional parameter setting of Extract. See About Procedural Replication for details. In the unlikely case that there is no native support, no support by fetching and no procedural replication support, there is no Oracle GoldenGate support.

Detailed support information for Oracle data types, objects, and operations starts with Details of Support for Objects and Operations in Oracle DML.

Extract Redo Support:

The following data types allow capturing directly from the redo logs and do not require any fetching. If used in a downstream mining configuration, the `NOUSERID` parameter may be used.

- `NUMBER`, `BINARY FLOAT`, `BINARY DOUBLE`, and (logical) `UROWID`
- `DATE` and `TIMESTAMP`
- `CHAR`, `VARCHAR2`, `LONG`, `NCHAR`, and `NVARCHAR2`, `BOOLEAN`
- `RAW`, `LONG RAW`, `CLOB`, `NCLOB`, `BLOB`, `SECUREFILE`, `BASICFILE`, and `BFILE` (LOB size limited to 4GB)
- `XML` columns stored as `CLOB`, `Binary` and Object-Relational (OR)
- `XMLType` columns and `XMLType` tables stored as `XML CLOB`, `XML` Object Relational, and `XML Binary`
- Native `JSON` datatype identified by the `DTYJSON` code.
- `UDTs` (user-defined or abstract data types) on `BYTE` semantics with source database compatibility 12.0.0.0.0 or higher
- `ANYDATA` data type with source database compatibility 12.0.0.0.0 or higher
- Hierarchy-enabled tables are managed by the Oracle XML database repository with source database compatibility 12.2.0.0.0 or higher and enabled procedural replication
- `REF` types with source database compatibility 12.2.0.0.0 or higher
- `DICOM` with source database compatibility 12.0.0.0.0 or higher
- `SDO _TOPO_GEOMETRY`, `SDO_GEORASTER`, or `ST_GEOMETRY` with source database compatibility 12.2.0.0.0 or higher and enabled procedural replication
- Identity columns with source database compatibility 18.1.0.0.0 or higher
- `SDO_RDF_TRIPLE_S` with source database compatibility 19.1.0.0.0 or higher

Data Types Fetched from the Database

Data types listed here are not readable in the redo logs and must be fetched by the Extract process during it's processing. The method for fetching these records is controlled by the use of the `FETCHOPTIONS` parameter.

It is recommended that the database that is generating the redo data is the same database that Oracle GoldenGate uses to fetch the data. However, if this is not possible, an Active Data Guard Standby database open for read-only can also be used as the fetch database.

`SECUREFILE` LOBs

- Modified with `DBMS_LOB.FRAGMENT_*` procedures
- `NOLOGGING` LOBs
- Deduplicated LOBs with a source database release less than 12gR2

UDTs that contain following data types:

- TIMESTAMP WITH TIMEZONE, TIMESTAMP WITH LOCAL TIMEZONE
- INTERVAL YEAR TO MONTH, INTERVAL DAY TO SECOND
- BINARY FLOAT, BINARY DOUBLE
- BFILE

Object tables contain the following attributes:

- Nested table
- SDO_TOP_GEOMETRY
- SDO_GEORASTER

Additional Considerations

- NUMBER can be up to the maximum size permitted by Oracle. The support of the range and precision for floating-point numbers depends on the host machine. In general, the precision is accurate to 16 significant digits, but you should review the database documentation to determine the expected approximations. Oracle GoldenGate rounds or truncates values that exceed the supported precision.
- Non-logical UROWID columns will be identified by Extract. A warning message is generated in the report file. The column information is not part of the trail record. All other supported datatypes of the record are part of the trail record and are replicated.
- TIMESTAMP WITH TIME ZONE as TZR (region ID) for initial loads, SQLEXEC or operations where the column can only be fetched from the database. In those cases, the region ID is converted to a time offset by the source database when the column is selected. Replicat applies the timestamp as date and time data into the target database with a time offset value.
- VARCHAR expansion from 4K to 32K (extended or long VARCHAR)
  - 32K long columns cannot be used as row identifiers:
    - Columns as part of a key or unique index
    - Columns in a KEYCOLS clause of the TABLE or MAP parameter.
  - 32K long columns as resolution columns in a CDR (conflict resolution and detection)
  - If an extended VARCHAR column is part of unique index or constraint, then direct path inserts to this table may cause Replicat to abend with a warning. Verify that the extended VARCHAR caused the abend by checking ALL_INDEXES or ALL_IND_COLUMNS for a unique index or ALL_CONS_COLUMNS or ALL_CONSTRAINTS for a unique constraint. Once you determine that an extended VARCHAR, you can temporarily drop the index or disable the constraint:
    - Unique Index: DROP INDEX *index_name*;
    - Unique Constraint: ALTER TABLE *table_name* MODIFY CONSTRAINT *constraint_name* DISABLE;
- BFILE column are replicating the locator. The file on the server file system outside of the database and is not replicated
- Multi-byte character data: The source and target databases must be logically identical in terms of schema definition for the tables and sequences being replicated. Transformation, filtering, and other manipulation cannot be used.

- The character sets between the two databases must be one of the following:
  - Identical on the source and on the target
  - Equivalent, which is not the same character set but containing the same set of characters
  - Target is a superset of the source

  Multi-byte data can be used in any semantics: bytes or characters.
- `UDTs` can have different source and target schemas. `UDTs`, including values inside object columns or rows, cannot be used within filtering criteria in `TABLE` or `MAP` statements, or as input or output for the Oracle GoldenGate column-conversion functions, `SQLEXEC`, or other built-in data manipulation tools. Support is only provided for like-to-like Oracle source and targets.

  To fully support object tables created using the `CREATE TABLE as SELECT (CTAS)` statement, Extract must be configured to capture DML from the CTAS statement. Oracle object table can be mapped to a non-Oracle object table in a supported target database.
- `XML` column type cannot be used for filtering and manipulation. You can map the `XML` representation of an object to a character column by means of a `COLMAP` clause in a `TABLE` or `MAP` statement.

  Oracle recommends the `AL32UTF8` character set as the database character set when working with `XML` data. This ensures the correct conversion by Oracle GoldenGate from source to target. With DDL replication enabled, Oracle GoldenGate replicates the CTAS statement and allows it to select the data from the underlying target tables. OIDs are preserved if `TRANLOGOPTIONS GETCTASDML` parameter is set. For `XMLType` tables, the row object IDs must match between source and target.
- For `JSON` datatype, `DTYJSON` is stored in the binary JSON format for query and space efficiency as well as transportability between platforms. A column with JSON data as text is declared using any of the text data types (`VARCHAR2`, `CLOB`) and the `IS JSON` constraint. JSON datatype is supported by Oracle GoldenGate Extract, and Replicat processes along with XStream Out, XStream In processes. JSON support limits the inline text JSON to 4K to prevent Replicat from abending.

  By default Extract writes native JSON columns in text format but using `binary_json_format` parameter forces to write in native format. So, this paramater must not be set for `VARCHAR2`, `NVARVAR2`, `CLOB`, `NCLOB`. The parameter is not set by default. If you are only replicating from Oracle to Oracle you can set the parameter and gain a bit of performance. Also the Column manipulation functions like `str` are supported only for text JSON.
- It is recommended that de-duplication is removed for LOB data types on the target database. If `DEDUPLICATION` is left enabled, it causes severe performance impact on the apply side.

SQLEXEC Limitations

There might be a few cases where replication support exists, but there are limitations of processing such as in case of using `SQLEXEC`. The following table lists these limitations:

Datatypes Supported By SQLEXEC	Support Limitations
`NUMBER, BINARY FLOAT, BINARY DOUBLE` `UROWID`	Special cases of: • XML types • UDTs • Object tables • Collections or nested tables
`(N)CHAR`, `(N) VARCHAR2 LONG`, `RAW`, `LONG RAW` `(N)CLOB`, `CLOB`, `BLOB`, `SECUREFILE`, `BASICFILE` and `BFILE`	Not supported
`XML` columns, `XMLType`	Not supported
Native `JSON` datatype	`VARCHAR2`, `NVARCHAR2`, `CLOB`, `NCLOB` not supported with the Extract parameter `binary_json_format`.
`UDT`	Not supported
`ANYDATA`	Not supported
Hierarchy-enabled tables	Not supported
`RET` Types	Not supported
`DICOM`	Not supported
`SDO_TOPO_GEOMETRY`, `SDO_GEORASTER`	Not supported
Identity columns	Not supported
`SDO_RDF_TRIPLE_S`	Not supported

Note:

`SECUREFILE` LOBs updated using `DBMS_LOG.FRAGMENT` or `SECUREFILE` LOBs that are set to `NOLOGGING` are fetched instead of read from the redo.

Note:

Any datatype not listed in the table is fully supported by SQLEXEC with the same limitations as the regular product.

- Handling Special Data Types
  It addresses special configuration requirements for different Oracle data types for Extract.
- Non-Supported Oracle Data Types

Handling Special Data Types

It addresses special configuration requirements for different Oracle data types for Extract.

Topics:.

- Multibyte Character Types
- Oracle Spatial Objects

- TIMESTAMP
- Large Objects (LOB)
- XML
- User Defined Types

Multibyte Character Types

Multi-byte characters are supported as part of a supported character set. If the semantics setting of an Oracle source database is `BYTE` and the setting of an Oracle target is `CHAR`, use the Replicat parameter `SOURCEDEFS` in your configuration, and place a definitions file that is generated by the `DEFGEN` utility on the target. These steps are required to support the difference in semantics, whether or not the source and target data definitions are identical. Replicat refers to the definitions file to determine the upper size limit for fixed-size character columns.

Oracle Spatial Objects

To replicate tables that contain one or more columns of `SDO_GEORASTER` object type from an Oracle source to an Oracle target, follow these instructions to configure Oracle GoldenGate to process them correctly.

1. Create a `TABLE` statement and a `MAP` statement for the georaster tables and also for the related raster data tables.
2. If the `METADATA` attribute of the `SDO_GEORASTER` data type in any of the values exceeds 1 MB, use the `DBOPTIONS` parameter with the `XMLBUFSIZE` option to increase the size of the memory buffer that stores the embedded `SYS.XMLTYPE` attribute of the `SDO_GEORASTER` data type. If the buffer is too small, Extract abends. See `XMLBUFSIZE` in *Reference for Oracle GoldenGate*.
3. To ensure the integrity of the target georaster tables and the spatial data, keep the trigger enabled on both source and target. Use the `REPERROR` option of the `MAP` parameter to handle the "ORA-01403 No data found" error that occurs as a result of keeping the trigger enabled on the target. It occurs when a row in the source georaster table is deleted, and the trigger cascades the delete to the raster data table. Both deletes are replicated. The replicated parent delete triggers the cascaded (child) delete on the target. When the replicated child delete arrives, it is redundant and generates the error. To use `REPERROR`, do the following:
   - Use a `REPERROR` statement in each `MAP` statement that contains a raster data table.
   - Use Oracle error 1403 as the SQL error.
   - Use any of the response options as the error handling.

A sufficient way to handle the errors on raster tables caused by active triggers on target georaster tables is to use `REPERROR` with `DISCARD` to discard the cascaded delete that triggers them. The trigger on the target georaster table performs the delete to the raster data table, so the replicated one is not needed.

```
MAP geo.st_rdt, TARGET geo.st_rdt, REPERROR (-1403, DISCARD) ;
```

If you need to keep an audit trail of the error handling, use `REPERROR` with `EXCEPTION` to invoke exceptions handling. For this, you create an exceptions table and map the source raster data table twice:

- once to the actual target raster data table (with `REPERROR` handling the 1403 errors).
- again to the exceptions table, which captures the 1403 error and other relevant information by means of a `COLMAP` clause.

For more information about using an exceptions table, see *Administering Oracle GoldenGate for Windows and UNIX*.

For more information about `REPERROR` options, see *Reference for Oracle GoldenGate*.

TIMESTAMP

To replicate timestamp data, Oracle Database normalizes `TIMESTAMP WITH LOCAL TIME ZONE` data to the local time zone of the database that receives it, the target database in case of Oracle GoldenGate. To preserve the original time stamp of the data that it applies, Replicat sets its session to the time zone of the source database. You can override this default and supply a different time zone by using the `SOURCETIMEZONE` parameter in the Replicat parameter file. To force Replicat to set its session to the target time zone, use the `PRESERVETARGETTIMEZONE` parameter.

To prevent Oracle GoldenGate from abending on `TIMESTAMP WITH TIME ZONE` as `TZR`, use the Extract parameter `TRANLOGOPTIONS` with `INCLUDEREGIONIDWITHOFFSET` to replicate `TIMESTAMP WITH TIMEZONE` as `TZR` from an Oracle source that is at least version 10g to an earlier Oracle target, or from an Oracle source to a non-Oracle target. This option allows replicating to Oracle versions that do not support `TIMESTAMP WITH TIME ZONE` as TZR and to database systems that only support time zone as a UTC offset.

You can also use the `SOURCETIMEZONE` parameter to specify the source time zone for data that is captured by an Extract that is earlier than version 12.1.2. Those versions do not write the source time zone to the trail.

Large Objects (LOB)

The following are some configuration guidelines for Extract LOBs.

1. Store large objects out of row if possible.
2. Extract captures LOBs from the redo log. For `UPDATE` operations on a LOB document, only the changed portion of the LOB is logged. To force whole LOB documents to be written to the trail when only the changed portion is logged, use the `TRANLOGOPTIONS` parameter with the `FETCHPARTIALLOB` option in the Extract parameter file. When Extract receives partial LOB content from the logmining server, it fetches the full LOB image instead of processing the partial LOB. Use this option when replicating to a non-Oracle target or in other conditions where the full LOB image is required.

XML

The following are tools for working with XML within Oracle GoldenGate constraints.

- Although Extract does not support the capture of changes made to an XML schema, you may be able to evolve the schemas and then resume replication of them without the need for a resynchronization, see Supporting Changes to XML Schemas.
- Extract captures XML from the redo log. For `UPDATE` operations on an XML document, only the changed portion of the XML is logged if it is stored as `OBJECT RELATIONAL` or `BINARY`. To force whole XML documents to be written to the trail when only the changed portion is logged, use the `TRANLOGOPTIONS` parameter with the `FETCHPARTIALXML` option in

the Extract parameter file. When Extract receives partial XML content from the logmining server, it fetches the full XML document instead of processing the partial XML. Use this option when replicating to a non-Oracle target or in other conditions where the full XML image is required.

User Defined Types

If Oracle Database is compatible with releases greater than or equal to 12.0.0.0.0, then Extract captures data from redo (no fetch), see Setting Flashback Query.

If replicating source data that contains user-defined types with the `NCHAR`, `NVARCHAR2`, or `NCLOB` attribute to an Oracle target, use the `HAVEUDTWITHNCHAR` parameter in the Replicat parameter file. When this type of data is encountered in the trail, `HAVEUDTWITHNCHAR` causes Replicat to connect to the Oracle target in `AL32UTF8`, which is required when a user-defined data type contains one of those attributes. `HAVEUDTWITHNCHAR` is required even if `NLS_LANG` is set to `AL32UTF8` on the target. By default Replicat ignores `NLS_LANG` and connects to an Oracle Database in the native character set of the database. Replicat uses the `OCIString` object of the Oracle Call Interface, which does not support `NCHAR`, `NVARCHAR2`, or `NCLOB` attributes, so Replicat must bind them as `CHAR`. Connecting to the target in `AL32UTF8` prevents data loss in this situation. `HAVEUDTWITHNCHAR` must appear before the `USERID` or `USERIDALIAS` parameter in the parameter file.

Non-Supported Oracle Data Types

Oracle GoldenGate does not support the following data types.

- Time offset values outside the range of +12:00 and -12:00..Oracle GoldenGate supports time offset values between +12:00 and -12:00.
- Tables that only contain a single column and that column one of the following:
  - UDT
  - LOB (CLOB, NCLOB, BLOB, BFILE)
  - XMLType column
  - VARCHAR2 (MAX) where the data is greater than 32KB
- Tables with LOB, UDT, XML, or XMLType column without one of the following:
  - Primary Key
  - Scalar columns with a unique constraint or unique index

  Table where the combination of all scalar columns do not guarantee uniqueness are unsupported.
- Tables with the following XML characteristics:
  - Tables with a primary key constraint made up of XML attributes
  - XMLType tables with a primary key based on an object identifier (PKOID).
  - XMLType tables, where the row object identifiers (OID) do not match between source and target
  - XMLType tables created by an empty CTAS statement.

  - XML schema-based XMLType tables and columns where changes are made to the XML schema (XML schemas must be registered on source and target databases with the `dbms_xml` package).
  - The maximum length for the entire `SET` value of an update to an XMLType larger than 32K, including the new content plus other operators and XQuery bind values.
  - SQL*Loader direct-path insert for XML-Binary and XML-OR.
- Tables with following UDT characteristics:
  - UDTs that contain CFILE or OPAQUE (except of XMLType)
  - UDTs with CHAR and VARCHAR attributes that contain binary or unprintable characters
  - UDTs using the RMTTASK parameter
- UDTs and nested tables with following condition:
  - Nested table UDTs with CHAR, NVARCHAR2 or NCLOB attributes.
  - Nested tables with CLOB, BLOB, extended (32k) VARCHAR2 or RAW attributes in UDTs.
  - Nested table columns/attributes that are part of any other UDT.
- When data in a nested table is updated, the row that contains the nested table must be updated at the same time. Otherwise there is no support.
- When VARRAYS and nested tables are fetched, the entire contents of the column are fetched each time, not just the changes. Otherwise there is no support.
- Object table contains the following attributes:
  - Nested table
  - SDO_TOPO_GEOMETRY
  - SDO_GEORASTER

See additional exclusions in Details of Support for Oracle Data Types and Objects.

Details of Support for Oracle Database Editions

This topic describes the Database Editions from the Oracle Database Product Family supported with the current Oracle GoldenGate release.

Oracle Database Express Edition (XE) is supported for delivery only and does not support any of the integrated features such as integrated Replicat or parallel Replicat in integrated mode.

Oracle Database Standard Edition 2 (SE2) is supported, with the following limitation:

- Extract, integrated Replicat, and parallel Replicat in integrated mode are limited to a single thread.

Oracle Database Enterprise Edition (EE) has full Oracle GoldenGate functionality.

Oracle Database Personal Edition (PE) is supported for delivery only, and does not support any of the integrated features such as integrated or parallel Replicat in integrated mode.

Details of Support for Objects and Operations in Oracle DML

This section outlines the Oracle objects and operations that Oracle GoldenGate supports for the capture and replication of DML operations.

Topics:

- Multitenant Container Database
- Tables, Views, and Materialized Views
- System Partitioning
- Sequences and Identity Columns
- Non-supported Objects and Operations in Oracle DML
- DML Auto Capture

Multitenant Container Database

Oracle GoldenGate captures from, and delivers to, a **multitenant container database**. See Configuring Oracle GoldenGate in a Multitenant Container Database .

Application Container are not supported.

Tables, Views, and Materialized Views

Oracle GoldenGate supports the following DML operations made to regular tables, index-organized tables, clustered tables, and materialized views.

- `INSERT`
- `UPDATE`
- `DELETE`
- Associated transaction control operations

Tip:

You can use the `DBA_GOLDENGATE_SUPPORT_MODE` data dictionary view to display information about the level of Oracle GoldenGate capture process support for the tables in your database. The `PLSQL` value of `DBA_GOLDENGATE_SUPPORT_MODE` indicates that the table is supported natively, but requires procedural supplemental logging. For more information, see the `DBA_GOLDENGATE_SUPPORT_MODE`. If you need to display all tables that have no primary and no non-null unique indexes, you can use the `DBA_GOLDENGATE_NOT_UNIQUE`. For more information, see `DBA_GOLDENGATE_NOT_UNIQUE`.

- Limitations of Support for Regular Tables

- Limitations of Support for Views
- Limitations of Support for Materialized Views
- Limitations of Support for Clustered Tables

Limitations of Support for Regular Tables

These limitations apply to Extract.

- Oracle GoldenGate supports tables that contain any number of rows.
- A row can be up to 4 MB in length. If Oracle GoldenGate is configured to include both the before and after image of a column in its processing scope, the 4 MB maximum length applies to the total length of the full before image plus the length of the after image. For example, if there are `UPDATE` operations on columns that are being used as a row identifier, the before and after images are processed and cannot exceed 4 MB in total. Before and after images are also required for columns that are not row identifiers but are used as comparison columns in conflict detection and resolution (CDR). Character columns that allow for more than 4 KB of data, such as a `CLOB`, only have the first 4 KB of data stored in-row and contribute to the 4MB maximum row length. Binary columns that allow for more than 4kb of data, such as a `BLOB` the first 8 KB of data is stored in-row and contributes to the 4MB maximum row length.
- Oracle GoldenGate supports the maximum number of columns per table that is supported by the database.
- Oracle GoldenGate supports the maximum column size that is supported by the database.
- Oracle GoldenGate supports tables that contain only one column, except when the column contains one of the following data types:
  - `LOB`
  - `LONG`
  - `LONG VARCHAR`
  - `Nested table`
  - User Defined Type (UDT)
  - `VARRAY`
  - `XMLType`
- Set `DBOPTIONS ALLOWUNUSEDCOLUMN` before you replicate from and to tables with unused columns.
- Oracle GoldenGate supports tables with these partitioning attributes:
  - Range partitioning
  - Hash Partitioning Interval Partitioning
  - Composite Partitioning
  - Virtual Column-Based Partitioning
  - Reference Partitioning
  - List Partitioning

- Oracle GoldenGate supports tables with virtual columns, but does not capture change data for these columns or apply change data to them: The database does not write virtual columns to the transaction log, and the Oracle Database does not permit DML on virtual columns. For the same reason, initial load data cannot be applied to a virtual column. You can map the data from virtual columns to non-virtual target columns.
- Oracle GoldenGate will not consider unique/index with virtual columns.
- Oracle GoldenGate supports replication to and from Oracle Exadata. To support Exadata Hybrid Columnar Compression, the source database compatibility must be set to 11.2.0.0.0 or higher.
- Oracle GoldenGate supports Transparent Data Encryption (TDE).
- Oracle GoldenGate supports `TRUNCATE` statements as part of its DDL replication support, or as standalone functionality that is independent of the DDL support.
- Oracle GoldenGate supports the capture of direct-load `INSERT`, with the exception of SQL*Loader direct-path insert for XML Binary and XML Object Relational. Supplemental logging must be enabled, and the database must be in archive log mode. The following direct-load methods are supported.
  - `/*+ APPEND */` hint
  - `/*+ PARALLEL */` hint
  - `SQLLDR` with `DIRECT=TRUE`
- Oracle GoldenGate fully supports capture from compressed objects for Extract.
- Oracle GoldenGate supports XA and PDML distributed transactions.
- Oracle GoldenGate supports DML operations on tables with `FLASHBACK ARCHIVE` enabled. However, Oracle GoldenGate does not support DDL that creates tables with the `FLASHBACK ARCHIVE` clause or DDL that creates, alters, or deletes the flashback data archive itself.

Limitations of Support for Views

These limitations apply to Extract.

- Oracle GoldenGate supports capture from a view when Extract is in initial-load mode (capturing directly from the source view, not the redo log).
- Oracle GoldenGate does not capture change data from a view, but it supports capture from the underlying tables of a view.
- Oracle GoldenGate can replicate to a view as long as the view is inherently updateable. The structures of the source tables and a target view must be identical.

Limitations of Support for Materialized Views

Materialized views are supported by Extract with the following limitations.

- Materialized views created `WITH ROWID` are not supported.
- The materialized view log can be created `WITH ROWID`.
- The source table must have a primary key.

- Truncates of materialized views are not supported. You can use a `DELETE FROM` statement.
- DML (but not DDL) from a full refresh of a materialized view is supported. If DDL support for this feature is required, open an Oracle GoldenGate support case.
- For Replicat the `Create MV` command must include the `FOR UPDATE` clause
- Either materialized views can be replicated or the underlying base table(s), but not both.

Limitations of Support for Clustered Tables

Indexed clusters are supported by Extract while hash clusters are not supported.

System Partitioning

System partitioning is an Oracle database feature that allows a table to be created with named partitions. A system partitioned table is not maintained by the database. Each DML must specify the partition where the row is to reside. Extract and all modes of Replicat support system partitioning. Each trail file record header pertaining to a system partitioned table includes the partition name. From Oracle GoldenGate 21c onward, a Partition Name Record (PNR) is generated for system partitioned tables, if it is included in the `PARTITION` parameter.

See `PARTITION | PARTITIONEXCLUDE` in the *Reference for Oracle GoldenGate*.

Sequences and Identity Columns

- Identity columns are supported from Oracle database 18c onward and requires Extract, Parallel Replicat in Integrated mode, or Integrated Replicat.
- Oracle GoldenGate supports the replication of sequence values and identity columns in a unidirectional and active-passive high-availability configuration.
- Oracle GoldenGate ensures that the target sequence values will always be higher than those of the source (or equal to them, if the cache is zero).
- Limitations of Support for Sequences

Limitations of Support for Sequences

These limitations apply to Extract.

- Oracle GoldenGate does not support the replication of sequence values in an active-active bi-directional configuration.
- The cache size and the increment interval of the source and target sequences must be identical. The cache can be any size, including 0 (`NOCACHE`).
- The sequence can be set to cycle or not cycle, but the source and target databases must be set the same way.
- Tables with default sequence columns are excluded from replication for Extract.

Non-supported Objects and Operations in Oracle DML

The following are additional Oracle objects or operations that are not supported by Extract:

- `REF` are supported natively for compatibility with Oracle Database 12.2 and higher, but not primary-key based `REFs` (`PKREFs`)
- Sequence values in an active-active bi-directional configuration
- Database Replay
- Tables created as `EXTERNAL`

DML Auto Capture

Oracle GoldenGate supports the following DML operations with auto capture mode:

- `TABLEEXCLUSION` parameter is supported.
- `TABLE` parameter is supported.
- Extract writes the table DML records delivered by the database for auto capture to trail file.

Details of Support for Objects and Operations in Oracle DDL

This topic outlines the Oracle objects and operation types that Oracle GoldenGate supports for the capture and replication of DDL operations.

Topics:

- Supported Objects and Operations in Oracle DDL
- Non-supported Objects and Operations in Oracle DDL

Supported Objects and Operations in Oracle DDL

DDL capture support is integrated into the database logmining server. You must set the database parameter compatibility to 11.2.0.4.0 or higher. Extract supports DDL that includes password-based column encryption, such as:

- `CREATE TABLE t1 (a number, b varchar2(32) ENCRYPT IDENTIFIED BY my_password);`
- `ALTER TABLE t1 ADD COLUMN c varchar2(64) ENCRYPT IDENTIFIED BY my_password;`

The following additional statements apply to Extract with respect to DDL support.

- All Oracle GoldenGate topology configurations are supported for Oracle DDL replication.
- Active-active (bi-directional) replication of Oracle DDL is supported between two (and only two) databases that contain identical metadata.
- Oracle GoldenGate supports DDL on the following objects:
  - clusters
  - directories
  - functions
  - indexes

  - packages
  - procedure
  - tables
  - tablespaces
  - roles
  - sequences
  - synonyms
  - triggers
  - types
  - views
  - materialized views
  - users
  - invisible columns
- Oracle Edition-Based Redefinition (EBR) database replication of Oracle DDL is supported for Extract for the following Oracle Database objects:
  - functions
  - library
  - packages (specification and body)
  - procedure
  - synonyms
  - types (specification and body)
  - views
- From Oracle GoldenGate 21c onward, DDLs that are greater than 4 MB in size will be provided replication support.
- Oracle GoldenGate supports Global Temporary Tables (GTT) DDL operations to be visible to Extract so that they can be replicated. You must set the `DDLOPTIONS` parameter to enable this operation because it is not set by default.
- Oracle GoldenGate supports dictionary for use with `NOUSERID` and `TRANLOGOPTIONS GETCTASDML`. This means that Extract receives object metadata from the LogMiner dictionary without querying the dictionary objects. Oracle GoldenGate uses the dictionary automatically when the source database compatibility parameter is greater than or equal to 11.2.0.4.

  When using dictionary and trail format in the Oracle GoldenGate release 12.2.*x*, Extract requires the Logminer patch to be applied on the mining database if the Oracle Database release is earlier than 12.1.0.2.
- Oracle GoldenGate supports replication of invisible columns in Extract. Trail format release 12.2 is required. Replicat must specify the `MAPINVISIBLECOLUMNS` parameter or explicitly map to invisible columns in the `COLMAP` clause of the `MAP` parameter.

  If `SOURCEDEFS` or `TARGETDEFS` is used, the metadata format of a definition file for Oracle tables must be compatible with the trail format. Metadata format 12.2 is compatible with trail format 12.2, and metadata format earlier than 12.2 is compatible with trail format earlier than 12.2. To specify the metadata format of a definition file, use the `FORMAT RELEASE` option of the `DEFSFILE` parameter when the definition file is generated in `DEFGEN`.

- DDL statements to create a namespace context (`CREATE CONTEXT`) are captured by Extract and applied by Replicat.
- Extract in pump mode supports the following DDL options:
  - `DDL INCLUDE ALL`
  - `DDL EXCLUDE ALL`
  - `DDL EXCLUDE OBJNAME`

  The `SOURCECATALOG` and `ALLCATALOG` option of `DDL EXCLUDE` is also supported.

  If no DDL parameter is specified, then all DDLs are written to trail. If `DDL EXCLUDE OBJNAME` is specified and the object owner is does not match an exclusion rule, then it is written to the trail.
- Starting with Oracle database 21c, the following DDL is available to support blocking of DML/DDL changes that are not replicated by Oracle GoldenGate:

  ```
  ALTER DATABASE [ENABLE | DISABLE] goldengate blocking mode;
  ```

  When Oracle GoldenGate blocking mode is enabled, DMLs that use `support_mode NONE` in tables and execute unsupported Oracle PL/SQL statements will fail with the following error:

  ```
  ORA-26981: "operation was unsupported during Oracle GoldenGate
  blocking mode"
  ```

  For Oracle database 21c, the following features cause a table to have `support_mode NONE` in Oracle GoldenGate:
  - `BFILE` as an attribute of ADT column, or typed table
  - Table with no scalars
  - OLAP AW$ table
  - Sharded queue table
  - Sorted Hash Cluster Table
  - Primary key constraint on ADT attribute in relational table
  - Primary key/unique key constraint on long `raw`/`varchar` (over 4000 bytes)
  - `V$DATABASE` column, `Goldengate_Blocking_Mode` can be queried to determine the current blocking mode status.
- For DDL auto capture mode:
  - It is relevant only for `DDL INCLUDE MAPPED` because Extract captures DDLs based on `TABLE` and `TABLEEXCLUDE` parameter.
  - Only table-related DDLs can be auto-captured.

  - DDLs to enable auto capture at table level:

    ```
    CREATE/ALTER TABLE … ENABLE LOGICAL REPLICATION ALLKEYS;
    ```

    or

    ```
    CREATE/ALTER TABLE … ENABLE LOGICAL REPLICATION ALLOW NOVALIDATE KEYS;
    ```

  See How to Capture Supplemental Logging for Oracle GoldenGate in *Oracle Database Utilities* guide.

- The following operations are supported for partition related DDLs and partition maintenance operations
  - Drop partition:

    If a partition is recreated with the same name, then it will get a new object number. The internal caches are cleared to minimize space consumption when a drop partition DDL is processed.
  - Truncate partition:

    Partition name and object number stays the same. Base table object version stays the same.
  - Rename partition:

    The partition object number stays the same but gets a new name. The base table's object version gets bumped. In memory name cache will get invalidated upon seeing this DDL and repopulated upon the next DML. The cache, which stores if a given partition object number is interesting or not will also need to be reevaluated as a the new partition name may switch from filtered to not filtered or vice versa.
  - Exchange partition:

    Exchanges data in a partition with that in a table or vice versa. The obj# of the partition being exchanged does not change. Dataobj# does change but is not used by Extract. The partition itself still belongs to the same table.
  - Merge partition:

    Merges one or more partitions into a new partition. The DDL creates the new partition and drops the partitions from which it was merged. In memory caches should be cleared to save space and the user should ensure proper filter rules for the newly created partition.
  - Split partition:

    The partition being split keeps its original name and object number and new partition is created for the split data. The user must ensure partition filter rules are correct for the newly created partition.
  - Coalesce partition:

    Reduces the number of partitions in a hash partitioned table. The specific partition that is coalesced is selected by the database, and is dropped after its contents have been redistributed. The remaining partitions keep their same name and object number. The internal caches should be cleared to minimize space consumption.
  - Modify partition:

Modifies default and real attributes of partitions, apart from adding or dropping of values for list partitions. All modifications leave the partitions name and object number intact.

- Move partition:

  Partition data is moved to a new tablespace. Partition name and number remain the same.

- Redef table:

  `dbms_redefinition` can be used to partition a table through the use of an interim table. The partitions are created on the interim table and after the `finish_redef` operation, the tables swap names. The partitions created on the interim table keep their names and object numbers when the tables are swapped. The Extract filter cache, needs to be reevaluated upon `finish_redef` as the partitions now belong to the base table. The user must ensure proper filter rules.

- Redef partition:

  When redefining a table, the partitions follow from the original table to the interim table. For example, consider the case where the original table has partitions, which live in the `USER` tablespace, and the interim table is created with no partitions and the table lives in the `NEW` tablespace. In this case, after the `finish_redef` operation, when the tables are swapped the partition still lives in the `USER` tablespace. Redef partition allows a partition to be moved to the interim table's `NEW` tablespace. The partition retains its name and object number.

- System generated partition names:

  When partitions are created automatically for hash partitions and operations such as split partition, the partition name is in the form of `SYS_P` *`sequence value`*. Similarly, subpartitions are of the form `SYS_SUBP` *`sequence value`*. It is recommended that the partition is renamed before excepting DML to conform to filter rules.

Non-supported Objects and Operations in Oracle DDL

Here's a list of non-supported objects and operations in Oracle DDL.

Topics:

- Excluded Objects
- Other Non-supported DDL

Excluded Objects

The following names or name prefixes are considered Oracle-reserved and must be excluded from the Oracle GoldenGate DDL configuration. Oracle GoldenGate will ignore objects that contain these names.

Excluded schemas:

```
"ANONYMOUS", // HTTP access to XDB
"APPQOSSYS", // QOS system user
"AUDSYS", // audit super user
"BI", // Business Intelligence
```

```
"CTXSYS", // Text
"DBSNMP", // SNMP agent for OEM
"DIP", // Directory Integration Platform
"DMSYS", // Data Mining
"DVF", // Database Vault
"DVSYS", // Database Vault
"EXDSYS", // External ODCI System User
"EXFSYS", // Expression Filter
"GSMADMIN_INTERNAL", // Global Service Manager
"GSMCATUSER", // Global Service Manager
"GSMUSER", // Global Service Manager
"LBACSYS", // Label Security
"MDSYS", // Spatial
"MGMT_VIEW", // OEM Database Control
"MDDATA",
"MTSSYS", // MS Transaction Server
"ODM", // Data Mining
"ODM_MTR", // Data Mining Repository
"OJVMSYS", // Java Policy SRO Schema
"OLAPSYS", // OLAP catalogs
"ORACLE_OCM", // Oracle Configuration Manager User
"ORDDATA", // Intermedia
"ORDPLUGINS", // Intermedia
"ORDSYS", // Intermedia
"OUTLN", // Outlines (Plan Stability)
"SI_INFORMTN_SCHEMA", // SQL/MM Still Image
"SPATIAL_CSW_ADMIN", // Spatial Catalog Services for Web
"SPATIAL_CSW_ADMIN_USR",
"SPATIAL_WFS_ADMIN", // Spatial Web Feature Service
"SPATIAL_WFS_ADMIN_USR",
"SYS",
"SYSBACKUP",
"SYSDG",
"SYSKM",
"SYSMAN", // Adminstrator OEM
"SYSTEM",
"TSMSYS", // Transparent Session Migration
"WKPROXY", // Ultrasearch
"WKSYS", // Ultrasearch
"WK_TEST",
"WMSYS", // Workspace Manager
"XDB", // XML DB
"XS$NULL",
"XTISYS", // Time Index
```

Special schemas:

```
"AURORA$JIS$UTILITY$", // JSERV
"AURORA$ORB$UNAUTHENTICATED", // JSERV
"DSSYS", // Dynamic Services Secured Web Service
"OSE$HTTP$ADMIN", // JSERV
"PERFSTAT", // STATSPACK
"REPADMIN",
"TRACESVR" // Trace server for OEM
```

Excluded tables (the * wildcard indicates any schema or any character):

```
"*.AQ$*", // advanced queues
"*.DR$*$*", // oracle text
"*.M*_*$$", // Spatial index
"*.MLOG$*", // materialized views
```

```
"*.OGGQT$*",
"*.OGG$*", // AQ OGG queue table
"*.ET$*", // Data Pump external tables
"*.RUPD$*", // materialized views
"*.SYS_C*", // constraints
"*.MDR*_*$", // Spatial Sequence and Table
"*.SYS_IMPORT_TABLE*",
"*.CMP*$*", // space management, rdbms >= 12.1
"*.DBMS_TABCOMP_TEMP_*", // space management, rdbms < 12.1
"*.MDXT_*$*" // Spatial extended statistics tables
```

Other Non-supported DDL

Oracle GoldenGate does not support the following:

- DDL on nested tables.
- DDL on identity columns.
- `ALTER DATABASE` and `ALTER SYSTEM` (these are not considered to be DDL) Using dictionary, you can replicate `ALTER DATABASE DEFAULT EDITION` and `ALTER PLUGGABLE DATABASE DEFAULT EDITION`. All other `ALTER [PLUGABLE] DATABASE` commands are ignored.
- DDL on a standby database.
- Database link DDL.
- DDL that creates tables with the `FLASHBACK ARCHIVE` clause and DDL that creates, alters, or deletes the flashback data archive itself. DML on tables with `FLASHBACK ARCHIVE` is supported.
- Some DDL will generate system generated object names. The names of system generated objects may not always be the same between two different databases. So, DDL operations on objects with system generated names should only be done if the name is exactly the same on the target.

A

Configuring the Initial Load for Classic Architecture

Oracle GoldenGate supports theses load methods in this section specifically for Oracle Database.

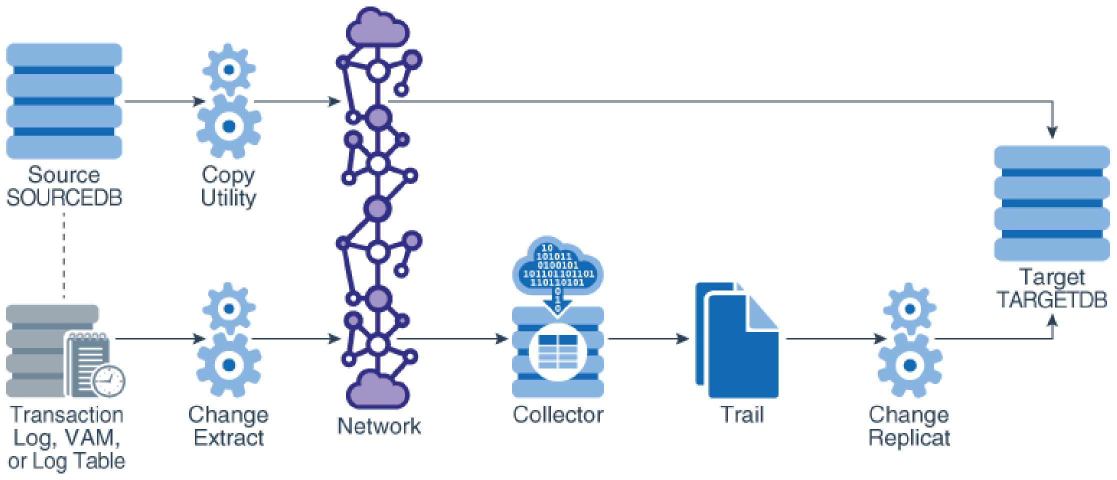
Select a method and follow its configuration steps to create the load processes and parameter files. To work with parameter files, see Using Oracle GoldenGate Parameter Files in *Administering Oracle GoldenGate*.

- Configuring a Load with an Oracle Data Pump
- Configuring a Direct Bulk Load to SQL*Loader
- Configuring a Load from an Input File to SQL*Loader

Configuring a Load with an Oracle Data Pump

This method uses the Oracle Data Pump utility to establish the target data. You start Extract, the data pumps, and Replicat at the SCN at which the copy stopped. Transactions that were included in the copy are skipped to avoid collisions from integrity violations. From the process start point, Oracle GoldenGate maintains data synchronization.

No initial-load Oracle GoldenGate processes are required for this method.

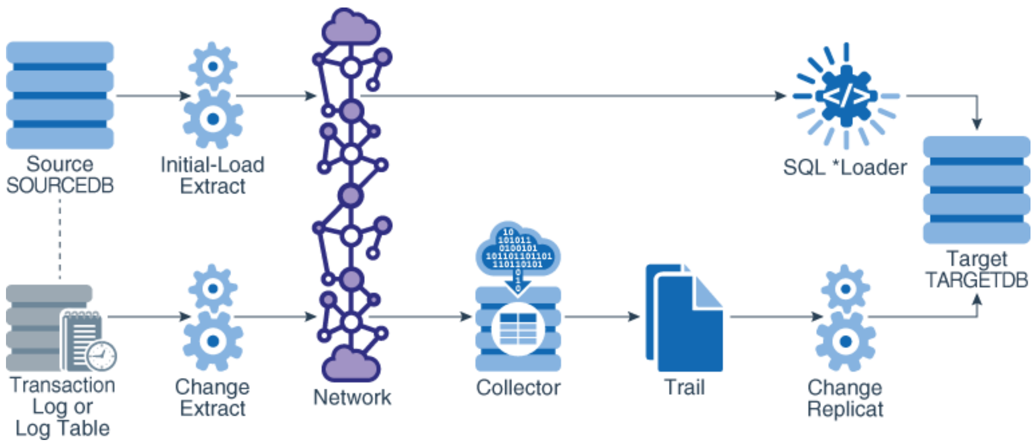


Configuring a Direct Bulk Load to SQL*Loader

With this method, you configure and run an Oracle GoldenGate initial-load Extract to extract complete source records and send them directly to an initial-load Replicat task. The initial-load Replicat task communicates with SQL*Loader to load data as a direct-path bulk load. Data mapping and transformation can be done by either the initial-load Extract or initial-load Replicat, or both. During the load, the change-synchronization groups that you configured in

#unique_148 and Configuring Oracle GoldenGate Replicat replicate incremental changes, which are then reconciled with the results of the load.

The following diagram shows configuring a direct bulk load to SQL*Loader.



Limitations:

- This method does not support extraction of `LOB` or `LONG` data. As an alternative, see Performing Instantiation From an Input File to SQL*Loader.
- This method does not support materialized views that contain `LOB`s, regardless of their size. It also does not support data encryption.

To Configure a Direct Bulk Load to SQL*Loader

1. Grant `LOCK ANY TABLE` to the Replicat database user on the target Oracle Database.
2. On the source and target systems, run GGSCI.
3. Start Manager on both systems.

   ```
   START MANAGER
   ```

4. On the source system, create the initial-load Extract.

   ```
   ADD EXTRACT initial-load_Extract, SOURCEISTABLE
   ```

   Where:

   - *`initial-load_Extract`* is the name of the initial-load Extract, up to eight characters.
   - `SOURCEISTABLE` directs Extract to read complete records directly from the source tables.

5. On the source system, create the initial-load Extract parameter file.

   ```
   EDIT PARAMS initial-load_Extract
   ```

6. Enter the initial-load Extract parameters in the order shown, starting a new line for each parameter statement. This example shows a three-part table name associated with a multitenant container database.

```
EXTRACT initext
USERIDALIAS tiger1
RMTHOST fin1, MGRPORT 7809 ENCRYPT AES192, KEYNAME securekey2
RMTTASK replicat, GROUP initrep
TABLE hq.hr.*;
```

Parameter	Description
`EXTRACT` *`initial-load_Extract`*	Specifies the name of the initial-load Extract, as stated with `ADD EXTRACT`.
`USERIDALIAS` *`alias`*	Specifies the alias of the database login credential that is assigned to Extract. This credential must exist in the Oracle GoldenGate credential store. For more information, see Establishing Oracle GoldenGate Credentials
`RMTHOST` *`hostname`*`, MGRPORT` *`portnumber`*`[, ENCRYPT` *`algorithm`* `KEYNAME` *`keyname`*`]`	Specifies the target system, the port where Manager is running, and optional encryption of data across TCP/IP.
`RMTTASK REPLICAT, GROUP` *`initial-load_Replicat`*	Specifies the process type (must be `REPLICAT`) and the name of the initial-load Replicat. Directs Manager on the target system to dynamically start the initial-load Replicat as a one-time task.
`TABLE` *`[container.]schema.table;`*	Specifies the tables to capture. • *`container`* is the name of the pluggable database, if this is a multitenant container database. You can use the `SOURCECATALOG` parameter to specify a default pluggable database instead of using three-part names. • *`schema`* is the schema name. • *`table`* is the table name.

7. Save and close the file.
8. On the target system, create the initial-load Replicat.

   `ADD REPLICAT` *`initial-load Replicat`*`, SPECIALRUN`

   Where:

   - *`initial-load Replicat`* is the name of the initial-load Replicat task.
   - `SPECIALRUN` identifies the initial-load Replicat as a one-time task, not a continuous process.

9. On the target system, create the initial-load Replicat parameter file.

   `EDIT PARAMS` *`initial-load Replicat`*

10. Enter the initial-load Replicat parameters in the order shown, starting a new line for each parameter statement. This example shows a three-part source table name associated with a multitenant container database.

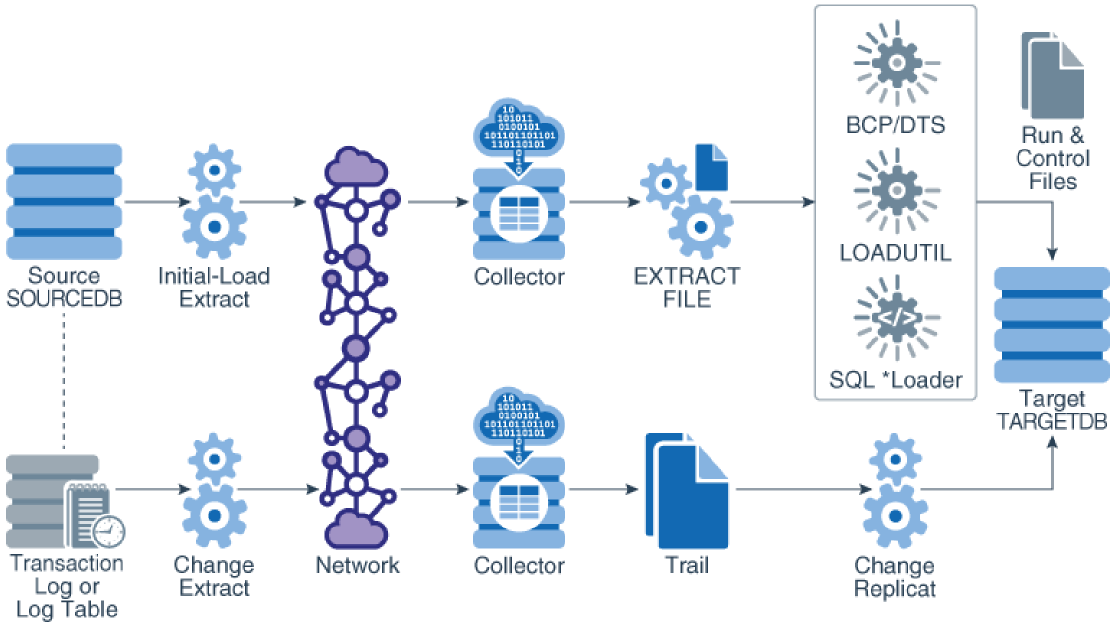
```
REPLICAT initrep
USERIDALIAS tiger2
BULKLOAD
```

```
ASSUMETARGETDEFS
MAP hq.hr.*, TARGET hr2.*;
```

Parameter	Description
`REPLICAT` *`initial-load Replicat`*	Specifies the name of the initial-load Replicat task, as stated with `ADD REPLICAT`.
`USERIDALIAS` *`alias`*	Specifies the alias of the database login credential that is assigned to Replicat. This credential must exist in the Oracle GoldenGate credential store.
`BULKLOAD`	Directs Replicat to interface directly with the Oracle SQL*Loader interface.
`ASSUMETARGETDEFS`	Assumes the source and target tables are identical, including semantics. If source and target definitions are different, you must create and specify a source-definitions file that both the change-synchronization and initial-load processes will use.
`MAP` *`[container.]schema.table`*`, TARGET` *`schema.table;`*	Specifies a relationship between a source and target table or tables. • If the source is a multitenant container database, *`container`* is the name of the pluggable database that contains the source objects specified with this `MAP` statement. You can use the `SOURCECATALOG` parameter to specify a default source pluggable database instead of using three-part names. • *`schema`* is the schema name. • *`table`* is the table name.

Configuring a Load from an Input File to SQL*Loader

With this method, an initial-load Extract extracts source records from the source tables and writes them to an extract file in external ASCII format. The files are read by SQL*Loader. During the load, the change-synchronization groups that you configured in Chapter 4 replicate incremental changes, which are then reconciled with the results of the load. As part of the load procedure, Oracle GoldenGate uses the initial-load Replicat to create run and control files required by the database utility. Any data transformation must be performed by the initial-load Extract on the source system because the control files are generated dynamically and cannot be pre-configured with transformation rules.



To Configure a Load from File to SQL*Loader

1. On the source and target systems, run GGSCI.
2. Start Manager on both systems.

```
START MANAGER
```

3. On the source system, create the initial-load Extract parameter file.

```
EDIT PARAMS initial-load Extract
```

4. Enter the initial-load Extract parameters in the order shown, starting a new line for each parameter statement. This example shows a three-part table name associated with a multitenant container database.

```
SOURCEISTABLE
USERIDALIAS tiger1
RMTHOST fin1, MGRPORT 7809 ENCRYPT AES192, KEYNAME securekey2
ENCRYPTTRAIL AES192
FORMATASCII, SQLLOADER
RMTFILE /ggs/dirdat/ie
TABLE hq.hr.*;
```

Parameter	Description
`SOURCEISTABLE`	Designates Extract as an initial load process that extracts records directly from the source tables.
`USERIDALIAS` *`alias`*	Specifies the alias of the database login credential that is assigned to Extract. This credential must exist in the Oracle GoldenGate credential store, see Establishing Oracle GoldenGate Credentials
`RMTHOST` *`hostname`*`,` `MGRPORT` *`portnumber`*`[,` `ENCRYPT` *`algorithm`* `KEYNAME` *`keyname`*`]`	Specifies the target system, the port where Manager is running, and optional encryption of data across TCP/IP.

Parameter	Description
`ENCRYPTTRAIL` *`algorithm`*	Encrypts the data in the remote file. For more information.
`FORMATASCII, SQLLOADER`	Produces a fixed-length, ASCII-formatted remote file that is compatible with SQL*Loader. This parameter must be listed before `RMTFILE`.
`RMTFILE` *`path`*	Specifies the absolute or full path name of an extract file that Extract creates and to which it writes the load data.
`TABLE` *`[container.]schema.table;`*	Specifies the tables to capture. • *`container`* is the name of the pluggable database, if this is a multitenant container database. You can use the `SOURCECATALOG` parameter to specify a default pluggable database instead of using three-part names. • *`schema`* is the schema name. • *`table`* is the table name.

5. Save and close the parameter file.
6. On the target system, create the initial-load Replicat parameter file.

```
EDIT PARAMS initial-load Replicat
```

7. Enter the initial-load Replicat parameters in the order shown, starting a new line for each parameter statement. This example shows a three-part source table name associated with a multitenant container database.

```
GENLOADFILES sqlldr.tpl
USERIDALIAS tiger2
EXTFILE /ggs/dirdat/ie
ASSUMETARGETDEFS
MAP hq.hr.*, TARGET hr2.*;
```

Parameter	Description
`GENLOADFILES` *`template`*	Generates run and control files for the database utility.
`USERIDALIAS` *`alias`*	Specifies the alias of the database login credential of the user that is assigned to Replicat. This credential must exist in the Oracle GoldenGate credential store, see Establishing Oracle GoldenGate Credentials
`EXTFILE` *`path`*	Specifies the extract file that you specified with the Extract parameter `RMTFILE`.
`ASSUMETARGETDEFS`	Assumes the source and target tables are identical, including semantics. If source and target definitions are different, you must create and specify a source-definitions file that both the change-synchronization and initial-load processes will use.
`MAP` *`[container.]schema.table`*`, TARGET` *`schema.table;`*	Specifies a relationship between a source and target table or tables. • If the source is a multitenant container database, *`container`* is the name of the pluggable database that contains the source objects specified with this `MAP` statement. You can use the `SOURCECATALOG` parameter to specify a default source pluggable database instead of using three-part names. • *`schema`* is the schema name. • *`table`* is the table name.

8. Save and close the parameter file.

B

Configuring the Data Pump Extract

A data pump can perform data filtering, mapping, and conversion, or it can be configured in pass-through mode, where data is passively transferred as-is, without manipulation.

These steps configure the data pump that reads the local trail and sends the data across the network to a remote trail. The data pump is optional, but recommended.

> **Note:**
>
> If you want to perform this task using microservices, see How to Add a Path in *Step by Step Data Replication Using Oracle GoldenGate Microservices Architecture*.

1. In GGSCI on the source system, create the data-pump parameter file.

   ```
   EDIT PARAMS name
   ```

   Where: *name* is the name of the data pump Extract.

2. Enter the data pump parameters in the order shown, starting a new line for each parameter statement. Your input variables will be different.

   Basic parameters for the data pump Extract group using two-part object names from a non-CDB database:

   ```
   EXTRACT extpump
   USERIDALIAS tiger1
   RMTHOST fin1, MGRPORT 7809 ENCRYPT AES192, KEYNAME securekey2
   RMTTRAIL /ggs/dirdat/rt
   SEQUENCE hr.employees_seq;
   TABLE hr.*;
   ```

   Basic parameters for the data pump Extract group using three-part object names from a trail that contains multitenant database data (including a pluggable database):

   ```
   EXTRACT extpump
   USERIDALIAS tiger1
   RMTHOST fin1, MGRPORT 7809 ENCRYPT AES192, KEYNAME securekey2
   RMTTRAIL /ggs/dirdat/rt
   TABLE test.ogg.tab1;
   SOURCECATALOG pdb1
   SEQUENCE hr.employees_seq;
   TABLE hr.*;
   SOURCECATALOG pdb2
   TABLE sales.*;
   TABLE acct.*;
   ```

Parameter	Description
`EXTRACT` *`group`*	*`group`* is the name of the data pump Extract. For more information, see *Reference for Oracle GoldenGate*.
`USERIDALIAS` *`alias`*	Specifies the alias of the database login credential of the user that is assigned to Extract. This credential must exist in the Oracle GoldenGate credential store.
`RMTHOST` *`hostname`*`,` `MGRPORT` *`portnumber`*`,` `[, ENCRYPT` *`algorithm`* `KEYNAME` *`keyname`*`]`	• `RMTHOST` specifies the name or IP address of the target system. • `MGRPORT` specifies the port number where Manager is running on the target. • `ENCRYPT` specifies optional encryption of data across TCP/IP.
`RMTTRAIL` *`pathname`*	Specifies the path name of the remote trail.
`SOURCECATALOG` *`container`*	Use this parameter when the source database is a multitenant container database. Specifies the name of a pluggable database that is to be used as the default container for all subsequent `TABLE` and `SEQUENCE` parameters that contain two-part names. This parameter enables you to use two-part object names (`schema.object`) rather than three-part names (`container.schema.object`). It remains in effect until another `SOURCECATALOG` parameter is encountered or a full three-part `TABLE` or `SEQUENCE` specification is encountered. Use this parameter when the source database is a multitenant container database.
`{TABLE | SEQUENCE}` *`[container.]schema.object;`*	Specifies a table or sequence, or multiple objects specified with a wildcard. In most cases, this listing will be the same as that in the primary Extract parameter file. • `TABLE` specifies a table or a wildcarded set of tables. • `SEQUENCE` specifies a sequence or a wildcarded set of sequences. • *`container`* is the name of the root container or pluggable database that contains the table or sequence, if this source database is a multitenant container database. See the `SOURCECATALOG` description in this table. • *`schema`* is the schema name or a wildcarded set of schemas. • *`object`* is the name of a table or sequence, or a wildcarded set of those objects. Terminate this parameter statement with a semi-colon. To exclude tables or sequences from a wildcard specification, use the `TABLEEXCLUDE` or `SEQUENCEEXCLUDE` parameter after the `TABLE` statement.

3. Enter any optional Extract parameters that are recommended for your configuration. You can edit this file at any point before starting processing by using the `EDIT PARAMS` command in GGSCI.
4. Save and close the file.

- Add the Data Pump Extract Group
  These steps add the data pump that reads the local trail and sends the data to the target.

Add the Data Pump Extract Group

These steps add the data pump that reads the local trail and sends the data to the target.

In GGSCI on the source system, issue the `ADD EXTRACT` command.

```
ADD EXTRACT group name, EXTTRAILSOURCE trail name
```

Where:

- group name is the name of the Extract group.
- `EXTTRAILSOURCE` *`trail name`* is the relative or fully qualified name of the local trail.

Example B-1

```
ADD EXTRACT financep, EXTTRAILSOURCE c:\ggs\dirdat\lt
```

C

Optional Parameters for Integrated Modes

This appendix contains optional parameters that may be required when operating Extract or Replicat in integrated Replicat mode.
Topics:

- Additional Parameter Options for Extract
  This section contains additional parameters that may be required for your Extract configuration.
- Additional Parameter Options for Integrated Replicat
  You can set these parameters by using the `DBOPTIONS` parameter with the `INTEGRATEDPARAMS` option or dynamically by issuing the `SEND REPLICAT` command with the `INTEGRATEDPARAMS` option in GGSCI.

Additional Parameter Options for Extract

This section contains additional parameters that may be required for your Extract configuration.

Extract uses a database logmining server in the mining database to mine the redo stream of the source database. You can set parameters that are specific to the logmining server by using the `TRANLOGOPTIONS` parameter with the `INTEGRATEDPARAMS` option in the Extract parameter file.

Note:

For detailed information and usage guidance for these parameters, see the "`DBMS_CAPTURE_ADM`" section in *Oracle Database PL/SQL Packages and Types Reference*.

The following parameters can be set with `INTEGRATEDPARAMS`:

- `CAPTURE_IDKEY_OBJECTS`: Controls the capture of objects that can be supported by `FETCH`. The default for Oracle GoldenGate is `Y` (capture ID key logical change records).
- `DOWNSTREAM_REAL_TIME_MINE`: Controls whether the logmining server operates as a real-time downstream capture process or as an archived-log downstream capture process. The default is `N` (archived-log mode). Specify this parameter to use real-time capture in a downstream logmining server configuration. For more information on establishing a downstream mining configuration, see Configuring a Downstream Mining Database .
- `INLINE_LOB_OPTIMIZATION`: Controls whether LOBs that can be processed inline (such as small LOBs) are included in the LCR directly, rather than sending LOB chunk LCRs. The default for Oracle GoldenGate is `Y` (Yes).
- `MAX_SGA_SIZE`: Controls the amount of shared memory used by the logmining server. The shared memory is obtained from the streams pool of the SGA. The default is 1 GB.

- `PARALLELISM`: Controls the number of processes used by the logmining server. The default is 2. For Oracle Standard Edition, this must be set to `1`.
- `TRACE_LEVEL`: Controls the level of tracing for the Extract logmining server. For use only with guidance from Oracle Support. The default for Oracle GoldenGate is `0` (no tracing).
- `WRITE_ALERT_LOG`: Controls whether the Extract logmining server writes messages to the Oracle alert log. The default for Oracle GoldenGate is `Y` (Yes).

See Managing Server Resources.

Additional Parameter Options for Integrated Replicat

You can set these parameters by using the `DBOPTIONS` parameter with the `INTEGRATEDPARAMS` option or dynamically by issuing the `SEND REPLICAT` command with the `INTEGRATEDPARAMS` option in GGSCI.

The default Replicat configuration as directed in Configuring Oracle GoldenGate Replicat should be sufficient. However, if needed, you can set the following inbound server parameters to support specific requirements.

Note:

For detailed information and usage guidance for these parameters, see the "`DBMS_APPLY_ADM`" section in *Oracle Database PL/SQL Packages and Types Reference*.

See *Reference for Oracle GoldenGate* for more information about the `DBOPTIONS` parameter.

- `COMMIT_SERIALIZATION`: Controls the order in which applied transactions are committed and has 2 modes, `DEPENDENT_TRANSACTIONS` and `FULL`. The default mode for Oracle GoldenGate is `DEPENDENT_TRANSACTIONS` where dependent transactions are applied in the correct order though may not necessarily be applied in source commit order. In `FULL` mode, the source commit order is enforced when applying transactions.
- `BATCHSQL_MODE`: Controls the batch execution scheduling mode including pending dependencies. A pending dependency is a dependency on another transaction that has already been scheduled, but not completely executed. The default is `DEPENDENT`. You can use following three modes:

  `DEPENDENT`

  Dependency aware scheduling without an early start. Batched transactions are scheduled when there are no pending dependencies.

  `DEPENDENT_EAGER`

  Dependency aware batching with early start. Batched transactions are scheduled irrespective of pending dependencies.

SEQUENTIAL

Sequential batching. Transactions are batched by grouping the transactions sequentially based on the original commit order.

- `DISABLE_ON_ERROR`: Determines whether the apply server is disabled or continues on an unresolved error. The default for Oracle GoldenGate is `N` (continue on errors), however, you can set the option to Y if you need to disable the apply server when an error occurs.
- `EAGER_SIZE`: Sets a threshold for the size of a transaction (in number of LCRs) after which Oracle GoldenGate starts applying data before the commit record is received. The default for Oracle GoldenGate is `15100`.
- `ENABLE_XSTREAM_TABLE_STATS`: Controls whether statistics on applied transactions are recorded in the `V$GOLDENGATE_TABLE_STATS` view or not collected at all. The default for Oracle GoldenGate is `Y` (collect statistics).
- `MAX_PARALLELISM`: Limits the number of apply servers that can be used when the load is heavy. This number is reduced again when the workload subsides. The automatic tuning of the number of apply servers is effective only if `PARALLELISM` is greater than 1 and `MAX_PARALLELISM` is greater than `PARALLELISM`. If `PARALLELISM` is equal to `MAX_PARALLELISM`, the number of apply servers remains constant during the workload. The default for Oracle GoldenGate is 50.
- `MAX_SGA_SIZE`: Controls the amount of shared memory used by the inbound server. The shared memory is obtained from the streams pool of the SGA. The default for Oracle GoldenGate is `INFINITE`.
- `MESSAGE_TRACKING_FREQUENCY`: Controls how often LCRs are marked for high-level LCR tracing through the apply processing. The default value is `2000000`, meaning that every 2 millionth LCR is traced. A value of zero (`0`) disables LCR tracing.
- `PARALLELISM`: Sets a minimum number of apply servers that can be used under normal conditions. Setting `PARALLELISM` to 1 disables apply parallelism, and transactions are applied with a single apply server process. The default for Oracle GoldenGate is `4`. For Oracle Standard Edition, this must be set to `1`.
- `PARALLELISM_INTERVAL`: Sets the interval in seconds at which the current workload activity is computed. Replicat calculates the mean throughput every 5 X `PARALLELISM_INTERVAL` seconds. After each calculation, the apply component can increase or decrease the number of apply servers to try to improve throughput. If throughput is improved, the apply component keeps the new number of apply servers. The parallelism interval is used only if `PARALLELISM` is set to a value greater than one and the `MAX_PARALLELISM` value is greater than the `PARALLELISM` value. The default is 5 seconds.
- `PRESERVE_ENCRYPTION`: Controls whether to preserve encryption for columns encrypted using Transparent Data Encryption. The default for Oracle GoldenGate is `N` (do not apply the data in encrypted form).
- `TRACE_LEVEL`: Controls the level of tracing for the Replicat inbound server. For use only with guidance from Oracle Support. The default for Oracle GoldenGate is `0` (no tracing).
- `WRITE_ALERT_LOG`: Controls whether the Replicat inbound server writes messages to the Oracle alert log. The default for Oracle GoldenGate is `Y` (yes).

D

Supporting Changes to XML Schemas

This appendix contains instructions for supporting changes to an XML schema. Extract does not support the capture of changes made to an XML schema.
Topics:

- Supporting RegisterSchema
  `RegisterSchema` can be handled by registering the schema definition on both source and target databases before any table is created that references the XML schema.
- Supporting DeleteSchema
  Issue `DeleteSchema` on the source database first.
- Supporting CopyEvolve
  The CopyEvolve procedure evolves, or changes, a schema and can modify tables by adding or removing columns.

Supporting RegisterSchema

`RegisterSchema` can be handled by registering the schema definition on both source and target databases before any table is created that references the XML schema.

Supporting DeleteSchema

Issue `DeleteSchema` on the source database first.

Once Replicat is caught up with the changes made to the source database, issue the `DeleteSchema` call on the target database.

Supporting CopyEvolve

The CopyEvolve procedure evolves, or changes, a schema and can modify tables by adding or removing columns.

The CopyEvolve procedure can also be used to change whether or not XML documents are valid. Handling `CopyEvolve` requires more coordination. Use the following procedure if you are issuing `CopyEvolve` on the source database.

1. Quiesce changes to dependent tables on the source database.
2. Execute the `CopyEvolve` on the primary or source database.
3. Wait for Replicat to finish applying all of the data from those tables to the target database.
4. Stop Replicat.
5. Apply the `CopyEvolve` on the target database.
6. Restart Replicat.

E

Preparing DBFS for an Active-Active Configuration

This appendix contains steps to configure Oracle GoldenGate to function within an active-active bidirectional or multi-directional environment where Oracle Database File System (DBFS) is in use on both (or all) systems.
Topics:

Supported Operations and Prerequisites

This topic lists what is supported by Oracle GoldenGate for DBFS.

Oracle GoldenGate for DBFS supports the following:

- Supported DDL (like `TRUNCATE` or `ALTER`) on DBFS objects except for `CREATE` statements on the DBFS objects. `CREATE` on DBFS must be excluded from the configuration, as must any schemas that will hold the created DBFS objects. The reason to exclude `CREATES` is that the metadata for DBFS must be properly populated in the SYS dictionary tables (which itself is excluded from Oracle GoldenGate capture by default).
- Capture and replication of DML on the tables that underlie the DBFS file system.

The procedures that follow assume that Oracle GoldenGate is configured properly to support active-active configuration. This means that it must be:

- Installed according to the instructions in this guide.
- Configured according to the instructions in the Oracle GoldenGate Windows and UNIX Administrator's Guide.

Applying the Required Patch

Apply the Oracle DBFS patch for bug-9651229 on both databases.

To determine if the patch is installed, run the following query:

```
connect / as sysdba
select  procedure_name
from    dba_procedures
where   object_name = 'DBMS_DBFS_SFS_ADMIN'
and procedure_name = 'PARTITION_SEQUENCE';
```

The query should return a single row. Anything else indicates that the proper patched version of DBFS is not available on your database.

Examples Used in these Procedures

The following procedures assume two systems and configure the environment so that DBFS users on both systems see the same DBFS files, directories, and contents that are kept in synchronization with Oracle GoldenGate.

It is possible to extend these concepts to support three or more peer systems.

Partitioning the DBFS Sequence Numbers

DBFS uses an internal sequence-number generator to construct unique names and unique IDs.

These steps partition the sequences into distinct ranges to ensure that there are no conflicts across the databases. After this is done, further DBFS operations (both creation of new file systems and subsequent file system operations) can be performed without conflicts of names, primary keys, or IDs during DML propagation.

1. Connect to each database as `sysdba`.

   Issue the following query on each database.

   ```
   select last_number
   from dba_sequences
   where sequence_owner = 'SYS'
   and sequence_name = 'DBFS_SFS_$FSSEQ'
   ```

2. From this query, choose the maximum value of `LAST_NUMBER` across both systems, or pick a high value that is significantly larger than the current value of the sequence on either system.

3. Substitute this value ("`maxval`" is used here as a placeholder) in both of the following procedures. These procedures logically index each system as `myid=0` and `myid=1`.

   Node1

   ```
   declare
   begin
   dbms_dbfs_sfs_admin.partition_sequence(nodes => 2, myid => 0, newstart
   => :maxval);
   commit;
   ```

```
end;
/
```

Node 2

```
declare
begin
dbms_dbfs_sfs_admin.partition_sequence( nodes => 2, myid => 1, newstart
=> :maxval);
commit;
end;
/
```

> **Note:**
>
> Notice the difference in the value specified for the `myid` parameter. These are the different index values.

For a multi-way configuration among three or more databases, you could make the following alterations:

- Adjust the maximum value that is set for `maxval` upward appropriately, and use that value on all nodes.
- Vary the value of `myid` in the procedure from 0 for the first node, 1 for the second node, 2 for the third one, and so on.

4. (Recommended) After (and only after) the DBFS sequence generator is partitioned, create a new DBFS file system on each system, and use only these file systems for DML propagation with Oracle GoldenGate. See Configuring the DBFS file system.

> **Note:**
>
> DBFS file systems that were created before the patch for bug-9651229 was applied or before the DBFS sequence number was adjusted can be configured for propagation, but that requires additional steps not described in this document. If you must retain old file systems, open a service request with Oracle Support.

Configuring the DBFS file system

To replicate DBFS file system operations, use a configuration that is similar to the standard bi-directional configuration for DML.

Some guidelines to follow while configuring Oracle GoldenGate for DBFS are:

- Use matched pairs of identically structured tables.
- Allow each database to have write privileges to opposite tables in a set, and set the other one in the set to read-only. For example:
  - Node1 writes to local table `t1` and these changes are replicated to `t1` on Node2.
  - Node2 writes to local table `t2` and these changes are replicated to `t2` on Node1.

- On Node1, `t2` is read-only. On Node2, `t1` is read-only.

DBFS file systems make this kind of table pairing simple because:

- The tables that underlie the DBFS file systems have the same structure.
- These tables are modified by simple, conventional DML during higher-level file system operations.
- The DBFS ContentAPI provides a way of unifying the namespace of the individual DBFS stores by means of mount points that can be qualified as read-write or read-only.

The following steps create two DBFS file systems (in this case named `FS1` and `FS2`) and set them to be read-write or read, as appropriate.

1. Run the following procedure to create the two file systems. (Substitute your store names for `FS1` and `FS2`.)
2. Run the following procedure to give each file system the appropriate access rights. (Substitute your store names for `FS1` and `FS2`.)

   In this example, note that on Node 1, store `FS1` is read-write and store `FS2` is read-only, while on Node 2 the converse is true: store `FS1` is read-only and store `FS2` is read-write.

   Note also that the read-write store is mounted as *local* and the read-only store is mounted as *remote*. This provides users on each system with an identical namespace and identical semantics for read and write operations. Local path names can be modified, but remote path names cannot.

Example E-1

```
declare
dbms_dbfs_sfs.createfile system('FS1');
dbms_dbfs_sfs.createfile system('FS2');

dbms_dbfs_content.registerStore('FS1',
'posix', 'DBMS_DBFS_SFS');
dbms_dbfs_content.registerStore('FS2',
'posix', 'DBMS_DBFS_SFS');
commit;
end;
/
```

Example E-2   Node 1

```
declare
dbms_dbfs_content.mountStore('FS1', 'local');
dbms_dbfs_content.mountStore('FS2', 'remote',
read_only => true);
commit;
end;
/
```

Example E-3   Node 2

```
declare
dbms_dbfs_content.mountStore('FS1', 'remote',
read_only => true);
dbms_dbfs_content.mountStore('FS2', 'local');
commit;
```

```
end;
/
```

Mapping Local and Remote Peers Correctly

The names of the tables that underlie the DBFS file systems are generated internally and dynamically.

Continuing with the preceding example, there are:

- Two nodes (Node 1 and Node 2 in the example).
- Four stores: two on each node (`FS1` and `FS2` in the example).
- Eight underlying tables: two for each store (a table and a ptable). These tables must be identified, specified in Extract `TABLE` statements, and mapped in Replicat `MAP` statements.

1. To identify the table names that back each file system, issue the following query. (Substitute your store names for `FS1` and `FS2`.)

   The output looks like the following examples.

2. Identify the tables that are *locally read-write* to Extract by creating the following `TABLE` statements in the Extract parameter files. (Substitute your pluggable database names, schema names, and table names as applicable.)
3. Link changes on each remote file system to the corresponding local file system by creating the following `MAP` statements in the Replicat parameter files. (Substitute your pluggable database, schema and table names.)

   This mapping captures and replicates local read-write *source* tables to remote read-only peer tables:

   - file system changes made to `FS1` on Node 1 propagate to `FS1` on Node 2.
   - file system changes made to `FS2` on Node 2 propagate to `FS2` on Node1.

   Changes to the file systems can be made through the DBFS ContentAPI (package `DBMS_DBFS_CONTENT`) of the database or through `dbfs_client` mounts and conventional file systems tools.

   All changes are propagated in both directions.

   - A user at the virtual root of the DBFS namespace on each system sees identical content.
   - For mutable operations, users use the `/local` sub-directory on each system.
   - For read operations, users can use either of the `/local` or `/remote` sub-directories, depending on whether they want to see local or remote content.

Example E-4

```
select fs.store_name, tb.table_name, tb.ptable_name
from table(dbms_dbfs_sfs.listTables) tb,
table(dbms_dbfs_sfs.listfile systems) fs
where   fs.schema_name = tb.schema_name
and fs.table_name = tb.table_name
and fs.store_name in ('FS1', 'FS2')
;
```

Example E-5   Example output: Node 1 (Your Table Names Will Be Different.)

```
STORE NAME     TABLE_NAME     PTABLE_NAME
-------------  -------------  -------------
FS1            SFS$_FST_100   SFS$_FSTP_100
FS2            SFS$_FST_118   SFS$_FSTP_118
```

Example E-6   Example output: Node 2 (Your Table Names Will Be Different.)

```
STORE NAME     TABLE_NAME     PTABLE_NAME
-------------  -------------  -------------
FS1            SFS$_FST_101   SFS$_FSTP_101
FS2            SFS$_FST_119   SFS$_FSTP_119
```

Example E-7   Node1

```
TABLE [container.]schema.SFS$_FST_100
TABLE [container.]schema.SFS$_FSTP_100;
```

Example E-8   Node2

```
TABLE [container.]schema.SFS$_FST_119
TABLE [container.]schema.SFS$_FSTP_119;
```

Example E-9   Node1

```
MAP [container.]schema.SFS$_FST_119, TARGET [container.]schema.SFS$_FST_118;
MAP [container.]schema.SFS$_FSTP_119, TARGET [container.]schema.SFS$_FSTP_118
```

Example E-10   Node2

```
MAP [container.]schema.SFS$_FST_100, TARGET
[container.]schema.SFS$_FST_101;MAP [container.]schema.SFS$_FSTP_100, TARGET
[container.]schema.SFS$_FSTP_101;
```