123
DBMS_LOGSTDBY

The DBMS_LOGSTDBY package provides subprograms for configuring and managing the logical standby database environment.

This chapter contains the following topics:

> **See Also:**
>
> *Oracle Data Guard Concepts and Administration*

DBMS_LOGSTDBY Overview

The DBMS_LOGSTDBY package helps you manage the SQL Apply (logical standby database) environment.

The subprograms in the DBMS_LOGSTDBY package help you to accomplish the following main objectives:

- Manage configuration parameters used by SQL Apply.

  For example, controlling how transactions are applied on the logical standby database, how much shared pool is used, and how many processes are used by SQL Apply to mine and apply the changes.
- Ensure an appropriate level of supplemental logging is enabled, and a LogMiner dictionary is built correctly for logical standby database creation.
- Provide a way to skip the application of changes to selected tables or entire schemas in the logical standby database, and specify ways to handle exceptions encountered by SQL Apply.
- Allow controlled access to tables in the logical standby database that may require maintenance.

DBMS_LOGSTDBY Security Model

You must have the DBA role to use the DBMS_LOGSTDBY package.

A prototype role, LOGSTDBY_ADMINISTRATOR, is created by default with RESOURCE and EXECUTE privileges on DBMS_LOGSTDBY. If you choose to use this role, consider granting ALTER DATABASE and ALTER SESSION privileges to the role so that the grantee can start and stop SQL Apply and can enable and disable the database guard.

The procedures associated with skipping transactions (SKIP and UNSKIP, SKIP_ERROR and UNSKIP_ERROR, and SKIP_TRANSACTION and UNSKIP_TRANSACTION) all require DBA privileges to execute because their scope may contain wildcard schemas. Oracle recommends that where SKIP procedures are specified, these be owned by a secure account with appropriate privileges on the schemas they act on (for example, SYS).

DBMS_LOGSTDBY Constants

The DBMS_LOGSTDBY package defines several enumerated constants for specifying parameter values. Enumerated constants must be prefixed with the package name, for example, DBMS_LOGSTDBY.SKIP_ACTION_SKIP.

The following table describes the constants for the proc_name parameter in the DBMS_LOGSTDBY.SKIP procedure.

Table 123-1 Constants for SKIP Options Flag

Constant	Description
MAX_EVENTS	Maximum number of events to log in the DBA_LOGSTDBY_EVENTS view. See the DBMS_LOGSTDBY APPLY_SET Procedure.
SKIP_ACTION_APPLY	Used inside the user-defined procedure registered with DBMS_LOGSTDBY.SKIP. Use this constant when setting the value of the SKIP_ACTION parameter with DBMS_LOGSTDBY_CONTEXT if you want SQL Apply to apply the DDL or PL/SQL statement.
SKIP_ACTION_ERROR	Used inside the user-defined procedure registered with DBMS_LOGSTDBY.SKIP. Use this constant when setting the value of the SKIP_ACTION parameter with DBMS_LOGSTDBY_CONTEXT if you want SQL Apply to error out.
SKIP_ACTION_REPLACE	Used inside the user-defined procedure registered with DBMS_LOGSTDBY.SKIP. Use this constant when setting the value of the SKIP_ACTION parameter with DBMS_LOGSTDBY_CONTEXT if you want SQL Apply to apply the replacement DDL. (This constant should not be used while handling an Oracle Supplied PL/SQL procedure call).
SKIP_ACTION_SKIP	Used inside the user-defined procedure registered with DBMS_LOGSTDBY.SKIP. Use this constant when setting the value of the SKIP_ACTION parameter with DBMS_LOGSTDBY_CONTEXT if you want SQL Apply to skip the associated DDL or Oracle supplied PL/SQL procedure call.

Summary of DBMS_LOGSTDBY Subprograms

This table describes each subprogram of the DBMS_LOGSTDBY procedure, including a reference to the section where each procedure is described in more detail.

In a multitenant container database (CDB), some subprograms must be called from the root. There may be other differences as well. See the individual subprogram descriptions for details.

Table 123-2 DBMS_LOGSTDBY Package Subprograms

Subprogram	Description
APPLY_SET Procedure	Sets the values of various parameters that configure and maintain SQL Apply.

Table 123-2 (Cont.) DBMS_LOGSTDBY Package Subprograms

Subprogram	Description
APPLY_UNSET Procedure	Restores the default values of various parameters that configure and maintain SQL Apply.
BUILD Procedure	Ensures supplemental logging is enabled properly and builds the LogMiner dictionary.
INSTANTIATE_TABLE Procedure	Creates and populates a table in the standby database from a corresponding table in the primary database.
IS_APPLY_SERVER Function	This function returns `TRUE` if it is executed from PL/SQL in the context of a logical standby apply server process. This function is used in conjunction with triggers that have the `fire_once` parameter in the `DBMS_DDL.SET_TRIGGER_FIRING_PROPERTY` subprogram set to `FALSE` (the default is `TRUE`). Such triggers are executed when the relevant target is updated by an apply process. This function can be used within the body of the trigger to ensure that the trigger takes different (or no) actions on the primary or on the standby.
MAP_PRIMARY_SCN Function	Maps an SCN relevant to the primary database to a corresponding SCN at the logical standby database. The mapped SCN is conservative in nature, and can thus be used to flash back the logical standby database to compensate for a flashback database operation performed at the primary database.
PREPARE_FOR_NEW_PRIMARY Procedure	Used after a failover, this procedure ensures a local logical standby database that was not involved in the failover has not processed more redo than the new primary database and reports the set of archive redo log files that must be replaced to ensure consistency
PURGE_SESSION Procedure	Identifies the archived redo log files that have been applied to the logical standby database and are no longer needed by SQL Apply
REBUILD Procedure	Records relevant metadata (including the LogMiner dictionary) in the redo stream in case a database that has recently changed its role to a primary database following a failover operation fails to do so during the failover process
SET_TABLESPACE Procedure	Moves metadata tables required by SQL Apply to the user-specified tablespace. By default, the metadata tables are created in the `SYSAUX` tablespace.
SKIP Procedure	Specifies rules that control database operations that should not be applied to the logical standby database
SKIP_ERROR Procedure	Specifies rules regarding what action to take upon encountering errors.
SKIP_TRANSACTION Procedure	Specifies transactions that should not be applied on the logical standby database. Be careful in using this procedure, because not applying specific transactions may cause data corruption at the logical standby database.
UNSKIP Procedure	Deletes rules specified by the `SKIP` procedure.
UNSKIP_ERROR Procedure	Deletes rules specified by the `SKIP_ERROR` procedure.
UNSKIP_TRANSACTION Procedure	Deletes rules specified by the `SKIP_TRANSACTION` procedure.

APPLY_SET Procedure

Use this procedure to set values of parameters that configure and manage SQL Apply in a logical standby database environment. All parameters, except for PRESERVE_COMMIT_ORDER, can be changed without having to stop SQL Apply.

In a CDB, the APPLY_SET procedure must be called from the root database.

Syntax

```
DBMS_LOGSTDBY.APPLY_SET (
     inname             IN VARCHAR,
     value              IN VARCHAR);
```

Parameters

Table 123-3 APPLY_SET Procedure Parameters

Parameter	Description
APPLY_SERVERS	Controls the number of APPLIER processes used to apply changes. The maximum number allowed is 1024, provided the MAX_SERVERS parameter is set to accommodate this.
EVENT_LOG_DEST	Controls where SQL Apply records the occurrence of an interesting event. It takes the following values: • DEST_ALL - All events will be recorded in the DBA_LOGSTDBY_EVENTS view and in the alert log. • DEST_EVENTS_TABLE - All events that contain information about user data will be recorded only in the DBA_LOGSTDBY_EVENTS view. This is the default value. For example, if SQL Apply receives an ORA-1403 error, the whole event is recorded in the DBA_LOGSTDBY_EVENTS view. Whereas, the alert log records only that SQL Apply stopped because of ORA-1403. No information regarding the user table or offending statement is logged in the alert log. However, if you stop the SQL Apply engine, it gets recorded in both the DBA_LOGSTDBY_EVENTS view and in the alert log. Note that this parameter affects the behavior of the following parameters: RECORD_APPLIED_DDL, RECORD_SKIP_DDL, RECORD_SKIP_ERRORS, and RECORD_UNSUPPORTED_OPERATIONS. For example, if RECORD_APPLIED_DDL is set to TRUE, but EVENT_LOG_DEST is set to DEST_EVENTS_TABLE, then the applied DDL string will only be recorded in the DBA_LOGSTDBY_EVENTS view.
LOG_AUTO_DEL_RETENTION_TARGET	This parameter setting is only meaningful if LOG_AUTO_DELETE has been set to TRUE. The value you supply for this parameter controls how long (in minutes) a remote archived log that is received from the primary database will be retained at the logical standby database once all redo records contained in the log have been applied at the logical standby database. The default value is 1440 minutes.

Table 123-3 (Cont.) APPLY_SET Procedure Parameters

Parameter	Description
LOG_AUTO_DELETE	Automatically deletes foreign archived redo log files as soon as they have been applied on the logical standby database. By default, a foreign archived redo log file is not deleted until 24 hours (the default value of LOG_AUTO_DEL_RETENTION_TARGET parameter) after it has been applied at the logical standby database. Set to TRUE to enable automatic deletion of archived redo log files. Set to FALSE to disable automatic deletion. The default value is TRUE.
MAX_EVENTS_RECORDED	Number of recent events that will be visible through the DBA_LOGSTDBY_EVENTS view. To record all events encountered by SQL Apply, use the DBMS_LOGSTDBY.MAX_EVENTS constant as the number value. The default value is 10,000.
MAX_SERVERS	Number of processes that SQL Apply uses to read and apply redo. The default value is 9. The maximum number allowed is 2048.
MAX_SGA	Number of megabytes from shared pool in System Global Area (SGA) that SQL Apply will use. The default value is 30 megabytes or one quarter of the value set for SHARED_POOL_SIZE, whichever is lower. The maximum size allowed is 4095 megabytes.
PREPARE_SERVERS	Controls the number of PREPARER processes used to prepare changes. The maximum number allowed is 1024, provided the MAX_SERVERS parameter is set to accommodate this.
PRESERVE_COMMIT_ORDER	TRUE: Transactions are applied to the logical standby database in the exact order in which they were committed on the primary database. This is the default parameter setting. FALSE: Transactions containing non-overlapping sets of rows may be committed in a different order than they were committed on the primary database. Regardless of the level chosen, modifications done to the same row are always applied in the same order as they happened on the primary database. See the Usage Notes for details and recommendations. You cannot modify this parameter while SQL Apply is running.
RECORD_APPLIED_DDL	Controls whether DDL statements that have been applied to the logical standby database are recorded in the location specified by the EVENT_LOG_DEST parameter. Specify one of the following values: TRUE: Indicates that DDL statements applied to the logical standby database are recorded in the DBA_LOGSTDBY_EVENTS table and the alert log. FALSE: Indicates that applied DDL statements are not recorded. This is the default parameter setting.
RECORD_SKIP_DDL	Controls whether skipped DDL statements are recorded in the location specified by the EVENT_LOG_DEST parameter. Specify one of the following values: TRUE: Skipped DDL statements are recorded in the DBA_LOGSTDBY_EVENTS table and the alert log. This is the default parameter setting. FALSE: Skipped DDL statements are not recorded in the DBA_LOGSTDBY_EVENTS table and the alert log.

Table 123-3 (Cont.) APPLY_SET Procedure Parameters

Parameter	Description
RECORD_SKIP_ERRORS	Controls whether skipped errors (as described by the SKIP_ERROR procedure) are recorded in the location specified by the EVENT_LOG_DEST parameter. Specify one of the following values: TRUE: Skipped errors are recorded in the DBA_LOGSTDBY_EVENTS table and the alert log. This is the default parameter setting. FALSE: Skipped errors are not recorded in the DBA_LOGSTDBY_EVENTS table and the alert log.
RECORD_UNSUPPORTED_OPERATIONS	Captures information about transactions running on the primary database that will not be supported by a logical standby database. This procedure records its information as events in the DBA_LOGSTDBY_EVENTS table. Specify one of the following values: TRUE: The information is captured and recorded as events in the DBA_LOGSTDBY_EVENTS table. FALSE: The information is not captured. This is the default.

If a parameter is changed while SQL Apply is running, the change will take effect at some point in the future. In such a case, an informational row is inserted into the DBA_LOGSTDBY_EVENTS view at the time the parameter change takes effect.

Additionally, if you are modifying a parameter while SQL Apply is running on an Oracle RAC configuration, you must be connected to the same instance where SQL Apply is running.

Exceptions

Table 123-4 APPLY_SET Procedure Exceptions

Exception	Description
ORA-16103	Logical Standby apply must be stopped to allow this operation
ORA-16104	invalid Logical Standby option requested
ORA-16236	Logical Standby metadata operation in progress

Usage Notes

- Use the APPLY_UNSET procedure to restore the default settings of a parameter.
- See *Oracle Data Guard Concepts and Administration* for help with tuning SQL Apply and for information about setting appropriate values for different parameters.

Examples

To record DDLs in the DBA_LOGSTDBY_EVENTS view and in the alert log, issue the following statement:

```
SQL> EXECUTE DBMS_LOGSTDBY.APPLY_SET('RECORD_APPLIED_DDL', TRUE);
```

APPLY_UNSET Procedure

Use the APPLY_UNSET procedure to restore the default values of the parameters that you changed with the APPLY_SET procedure.

In a CDB, the APPLY_UNSET procedure must be called from the root database.

Syntax

```
DBMS_LOGSTDBY.APPLY_UNSET (
     inname          IN VARCHAR);
```

Parameters

The parameter information for the APPLY_UNSET procedure is the same as that described for the APPLY_SET procedure. See Table 123-3 for complete parameter information.

Exceptions

Table 123-5 APPLY_UNSET Procedure Exceptions

Exception	Description
ORA-16103	Logical Standby apply must be stopped to allow this operation
ORA-16104	invalid Logical Standby option requested
ORA-16236	Logical Standby metadata operation in progress

Usage Notes

- Use the APPLY_SET procedure to specify a nondefault value for a parameter.

Examples

If you previously specified that applied DDLs show up in the DBA_LOGSTDBY_EVENTS view and the alert log, you can restore the default behavior of SQL Apply regarding applied DDL statements with the following statement:

```
SQL> EXECUTE DBMS_LOGSTDBY.APPLY_UNSET('RECORD_APPLIED_DDL');
```

BUILD Procedure

Use this procedure on the primary database to record relevant metadata (LogMiner dictionary) information in the redo log, which will subsequently be used by SQL Apply. This procedure will enable database-wide primary- and unique-key supplemental logging, if necessary.

In a CDB, the BUILD procedure must be called from the root database on the primary. Additionally, you cannot add or remove PDBs from a CDB while this procedure is executing.

> **Note:**
>
> In databases created using Oracle Database 11*g* release 2 (11.2) or later, supplemental logging information is automatically propagated to any existing physical standby databases. However, for databases in earlier releases, or if the database was created using an earlier release and then upgraded to 11.2, you must check whether supplemental logging is enabled at the physical standby(s) if it is also enabled at the primary database. If it is not enabled at the physical standby(s), then before performing a switchover or failover, you must enable supplemental logging on all existing physical standby databases. To do so, issue the following SQL command on each physical standby:
>
> ```
> SQL>  ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY, UNIQUE INDEX)
> COLUMNS;
> ```
>
> If you do not do this, then any logical standby that is also in the same Data Guard configuration will be unusable if a switchover or failover is performed to one of the physical standby databases. If a switchover or failover has already occurred and supplemental logging was not enabled, then you must recreate all logical standby databases.

Syntax

```
DBMS_LOGSTDBY.BUILD;
```

Usage Notes

- Supplemental log information includes extra information in the redo logs that uniquely identifies a modified row in the logical standby database, and also includes information that helps efficient application of changes to the logical standby database.
- LogMiner dictionary information allows SQL Apply to interpret data in the redo logs.
- `DBMS_LOGSTDBY.BUILD` should be run only once for each logical standby database you want to create. You do not need to use DBMS_LOGSTDBY.BUILD for each Oracle RAC instance.
- `DBMS_LOGSTDBY.BUILD` waits for all transactions (including distributed transactions) that are active at the time of the procedure invocation to complete before returning. See *Oracle Database Administrator's Guide* for information about how to handle in-doubt transactions.

Examples

To build the LogMiner dictionary in the redo stream of the primary database and to record additional information so that a logical standby database can be instantiated, issue the following SQL statement at the primary database

```
SQL> EXECUTE DBMS_LOGSTDBY.BUILD;
```

INSTANTIATE_TABLE Procedure

This procedure creates and populates a table in the standby database from a corresponding table in the primary database.

The table requires the name of the database link (`dblink`) as an input parameter. If the table already exists in the logical standby database, it will be dropped and re-created based on the

table definition at the primary database. This procedure only brings over the data associated with the table, and not the associated indexes and constraints.

Use the INSTANTIATE_TABLE procedure to:

- Add a table to a standby database.
- Re-create a table in a standby database.

In a CDB, the INSTANTIATE_TABLE procedure must be called from within the container in which the table to be instantiated resides. Additionally, the database link that is provided to the primary database must point to the corresponding container on the primary.

Syntax

```
DBMS_LOGSTDBY.INSTANTIATE_TABLE (
     schema_name         IN VARCHAR2,
     table_name          IN VARCHAR2,
     dblink              IN VARCHAR2);
```

Parameters

Table 123-6 INSTANTIATE_TABLE Procedure Parameters

Parameter	Description
schema_name	Name of the schema
table_name	Name of the table to be created or re-created in the standby database
dblink	Name of the database link account that has privileges to read and lock the table in the primary database, as well as the SELECT_CATALOG_ROLE on the primary database

Exceptions

Table 123-7 INSTANTIATE_TABLE Procedure Exceptions

Exception	Description
ORA-16103	Logical Standby apply must be stopped to allow this operation
ORA-16236	Logical Standby metadata operation in progress
ORA-16276	Specified database link does not correspond to primary database
ORA-16277	Specified table is not supported by logical standby database
ORA-16278	Specified table has a multi-object skip rule defined

Usage Notes

- Use this procedure to create and populate a table in a way that keeps the data on the standby database transactionally consistent with the primary database.
- This table will not be synchronized with the rest of the tables being maintained by SQL Apply and SQL Apply will not start to maintain it until SQL Apply encounters redo that occurred after the table was instantiated from the primary. The SCN at which the table was instantiated from the primary database is available in the DBA_LOGSTDBY_EVENTS view.
- The specified table must be a table that is supported by logical standby (that is, it does not appear in the DBA_LOGSTDBY_UNSUPPORTED_TABLES view on the primary database).

- If there are any skip rules that specifically name this table (without any wildcards), those skip rules will be dropped as part of `INSTANTIATE_TABLE`, so that the table will be properly maintained by SQL Apply in the future. If there are skip rules that indirectly reference this table (match a skip rule with a wildcard in the `schema_name` or `table_name`, and have a TABLE, DML, or SCHEMA_DDL statement type), `INSTANTIATE_TABLE` will fail with an ORA-16278 error. Any multi-object skip rules that pertain to the table must be dropped or changed before re-attempting the `INSTANTIATE_TABLE` call.

Examples

```
SQL> EXECUTE DBMS_LOGSTDBY.INSTANTIATE_TABLE (-
     SCHEMA_NAME => 'HR', TABLE_NAME => 'EMPLOYEES', -
     DBLINK => 'INSTANTIATE_TBL_LINK');
```

IS_APPLY_SERVER Function

This function returns `TRUE` if it is executed from PL/SQL in the context of a logical standby apply server process.

This function is used in conjunction with triggers that have the `fire_once` parameter in the `DBMS_DDL.SET_TRIGGER_FIRING_PROPERTY` subprogram set to `FALSE` (the default is `TRUE`). Such triggers are executed when the relevant target is updated by an apply process. This function can be used within the body of the trigger to ensure that the trigger takes different (or no) actions on the primary or on the standby.

Syntax

```
DBMS_LOGSTDBY.IS_APPLY_SERVER
RETURN BOOLEAN;
```

Parameters

None

MAP_PRIMARY_SCN Function

This function returns an SCN on the standby that predates the supplied SCN from the primary database by at least 5 minutes.

It can be used to determine a safe SCN to use in a compensating flashback database operation at the logical standby database, following a flashback database operation or a point-in-time recovery operation at the primary database.

Syntax

```
DBMS_LOGSTDBY.MAP_PRIMARY_SCN(primary_scn NUMBER) RETURN NUMBER;
```

Exceptions

Table 123-8 MAP_PRIMARY_SCN Function Exceptions

Exception	Description
ORA-20001	Primary SCN is before mapped range
ORA-20002	SCN mapping requires `PRESERVE_COMMIT_ORDER` to be `TRUE`

Usage Notes

Use this function to get a conservative SCN at the logical standby database that corresponds to an SCN at the primary database. This function is useful in the context of doing compensating flashback database operations at the logical standby following a flashback database or a point-in-time recovery operation done at the primary database.

PREPARE_FOR_NEW_PRIMARY Procedure

The `PREPARE_FOR_NEW_PRIMARY` procedure must be invoked at a logical standby database following a failover, if that standby database was not the target of the failover operation.

Such a standby database must process the exact same set of redo logs processed at the new primary database. This routine ensures that the local logical standby database has not processed more redo than the new primary database and reports the set of archive logs that must be replaced to ensure consistency. The set of replacement logs will be reported in the alert.log. These logs must be copied to the logical standby and registered using the `ALTER DATABASE REGISTER LOGICAL LOGFILE` statement.

In a CDB, the `PREPARE_FOR_NEW_PRIMARY` procedure must be called from the root database.

Syntax

```
DBMS_LOGSTDBY.PREPARE_FOR_NEW_PRIMARY (
           FORMER_STANDBY_TYPE         IN VARCHAR2,
           DBLINK                      IN VARCHAR2);
```

Parameters

Table 123-9 PREPARE_FOR_NEW_PRIMARY Procedure Parameters

Parameter	Description
`FORMER_STANDBY_TYPE`	The type of standby database that was the target of the failover operation to become the new primary database. Valid values are '`PHYSICAL`' if the new primary was formerly a physical standby, and '`LOGICAL`' if the new primary database was formerly a logical standby database.
`DBLINK`	The name of a database link to the new primary database

Exceptions

Table 123-10 PREPARE_FOR_NEW_PRIMARY Procedure Exceptions

Exception	Description
`ORA-16104`	Invalid Logical Standby option.
`ORA-16109`	Failed to apply log data from previous primary.

Usage Notes

- This routine is intended only for logical standby systems.This routine will fail if the new primary database was formerly a logical standby database and the LogMiner dictionary build has not completed successfully.Log files displayed in the alert log will be referred to as *terminal logs*. Users should keep in mind that file paths are relative to the new primary

database and may not resolve locally.Upon manual registration of the terminal logs, users should complete the process by calling either `START LOGICAL STANDBY APPLY` if the new primary database was formerly a physical standby database or `START LOGICAL STANDBY APPLY NEW PRIMARY` if the new primary database was formerly a logical standby database.See the alert log for more details regarding the reasons for any exception.

Examples

```
SQL> EXECUTE DBMS_LOGSTDBY.PREPARE_FOR_NEW_PRIMARY (  -
                FORMER_STANDBY_TYPE => 'LOGICAL',    -
                DBLINK => 'dblink_to_newprimary');
```

PURGE_SESSION Procedure

`PURGE_SESSION` identifies all archived redo log files that have been applied to the logical standby database and are no longer needed by SQL Apply.

Once identified, you can issue operating system commands to delete some or all of the unnecessary archived redo log files.

In a CDB, the `PURGE_SESSION` procedure must be called from the root database.

Syntax

```
DBMS_LOGSTDBY.PURGE_SESSION;
```

Exceptions

Table 123-11 PURGE_SESSION Procedure Exceptions

Exception	Description
ORA-01309	Invalid session

Usage Notes

- This procedure does not delete the archived redo log files. You must issue operating system commands to delete unneeded files.
- This procedure updates the `DBA_LOGMNR_PURGED_LOG` view that displays the archived redo log files that have been applied to the logical standby database.
- In Oracle Database 10*g* Release 2, metadata related to the archived redo log files (and the actual archived redo log files) are purged automatically based on the default setting of the `LOG_AUTO_DELETE` parameter described in the `DBMS_LOGSTDBY.APPLY_SET` procedure described.

Example

To identify and remove unnecessary files:

1. Enter the following statement on the logical standby database:

   ```
   SQL> EXECUTE DBMS_LOGSTDBY.PURGE_SESSION;
   ```

2. Query the `DBA_LOGMNR_PURGED_LOG` view to list the archived redo log files that can be removed:

   ```
   SQL> SELECT * FROM DBA_LOGMNR_PURGED_LOG;
   ```

```
FILE_NAME
   ----------------------------------
   /boston/arc_dest/arc_1_40_509538672.log
   /boston/arc_dest/arc_1_41_509538672.log
   /boston/arc_dest/arc_1_42_509538672.log
   /boston/arc_dest/arc_1_43_509538672.log
   /boston/arc_dest/arc_1_44_509538672.log
   /boston/arc_dest/arc_1_45_509538672.log
   /boston/arc_dest/arc_1_46_509538672.log
   /boston/arc_dest/arc_1_47_509538672.log
```

3. Use operating system-specific commands to delete archived redo log files from the file system.

REBUILD Procedure

This procedure is used if a database that has recently changed its role to a primary database following a failover operation fails to record relevant metadata (including the LogMiner dictionary) in the redo stream required for other logical standby databases.

In a CDB, the `REBUILD` procedure must be called from the root database.

Syntax

```
DBMS_LOGSTDBY.REBUILD;
```

Usage Notes

- LogMiner dictionary information is logged in the redo log files.The standby redo log files (if present) are archived.

Examples

```
SQL> EXECUTE DBMS_LOGSTDBY.REBUILD;
```

SET_TABLESPACE Procedure

This procedure moves metadata tables required by SQL Apply to the user-specified tablespace.

By default, the metadata tables are created in the `SYSAUX` tablespace. SQL Apply cannot be running when you invoke this procedure.

In a CDB, the `SET_TABLESPACE` procedure must be called from the root database.

Syntax

```
DBMS_LOGSTDBY.SET_TABLESPACE(
          NEW_TABLESPACE IN VARCHAR2)
```

Parameters

Table 123-12 SET_TABLE SPACE Procedure Parameters

Parameter	Description
NEW_TABLESPACE	Name of the new tablespace where metadata tables will reside.

Exceptions

Table 123-13 SET_TABLESPACE Procedure Exceptions

Exception	Description
ORA-16103	Logical Standby apply must be stopped to allow this operation
ORA-16236	Logical Standby metadata operation in progress

Examples

To move metadata tables to a new tablespace named LOGSTDBY_TBS, issue the following statement:

```
SQL> EXECUTE DBMS_LOGSTDBY.SET_TABLESPACE (new_tablespace => 'LOGSTDBY_TBS');
```

SKIP Procedure

The SKIP procedure can be used to define rules that will be used by SQL Apply to skip the application of certain changes to the logical standby database.

For example, the SKIP procedure can be used to skip changes to a subset of tables in the logical standby database. It can also be used to specify DDL statements that should not be applied at the logical standby database or should be modified before they are applied in the logical standby database. One reason why a DDL statement may need to be modified is to accommodate a different directory structure on the logical standby database.

> **Note:**
>
> For information about skipping containers, see "Skipping Containers."

Syntax

```
DBMS_LOGSTDBY.SKIP (
     stmt                      IN VARCHAR2,
     schema_name               IN VARCHAR2 DEFAULT NULL,
     object_name               IN VARCHAR2 DEFAULT NULL,
     proc_name                 IN VARCHAR2 DEFAULT NULL,
     use_like                  IN BOOLEAN DEFAULT TRUE,
     esc                       IN CHAR1 DEFAULT NULL);
```

Parameters

Table 123-14 SKIP Procedure Parameters

Parameter	Description
stmt	Either a keyword that identifies a set of SQL statements or a specific SQL statement. The use of keywords simplifies configuration since keywords, generally defined by the database object, identify all SQL statements that operate on the specified object. Table 123-15 shows a list of keywords and the equivalent SQL statements, either of which is a valid value for this parameter. The keyword PL/SQL is used for the execution of Oracle-supplied packages which are supported for replication.
schema_name	The name of one or more schemas (wildcards are permitted) associated with the SQL statements identified by the stmt parameter. If not applicable, this value must be set to NULL.
object_name	The name of one or more objects (wildcards are permitted) associated with the SQL statements identified by the stmt. If not applicable, this value must be set to NULL.
proc_name	Name of a stored procedure to call when SQL Apply determines that a particular statement matches the filter defined by the stmt, schema_name, and object_name parameters. Specify the procedure in the following format: 'schema.package.procedure' This procedure returns a value that directs SQL Apply to perform one of the following: execute the statement, skip the statement, or execute a replacement statement. The procedures to be invoked in the case of DDL or PL/SQL take no arguments. You can access the various information needed inside the procedure by accessing the context associated with the namespace,LSBY_APPLY_CONTEXT. For a full list of parameters that are accessible in the context of the skip procedure, see the DBMS_LOGSTBDY_CONTEXT package. The parameters of interest in the case of DDLs are: STATEMENT, STATEMENT_TYPE, SCHEMA, NAME, CURENT_SCHEMA, XIDUSN, XIDSLT, XIDSQN and SKIP_ACTION. The parameters of interest in the case of PL/SQL are: STATEMENT, PACKAGE_SCHMEMA, PACKAGE_NAME, PROCEDURE_NAME, CURRENT_SCHEMA, XIDUSN, XIDSLT, XIDSQN, EXIT_STATUS, and SKIP_ACTION.. **Note 1**: The DBMS_LOGSTDBY.SKIP_ACTION_REPLACE constant is not supported for PL/SQL. **Note 2**: SQL Apply calls the skip handler when the procedure's exit is processed. **Note 3**: The use_like parameter must be set to FALSE for PL/SQL since wildcarding PL/SQL is not supported.

Table 123-14 (Cont.) SKIP Procedure Parameters

Parameter	Description
proc_name (cont.)	A sample conditional skip rule on DBMS_RLS.DROP_POLICY is as follows:

```
create or replace procedure sec.mgr.skip_drop_policy
is
l_stmt  CLOB;
l_pkgown varchar2(30);
l_pkgnam varchar2(30);
l_procnm varchar2(30);
l_cur_schema varchar2(30);
l_xidusn   number;
l_xidslt   number;
l_xidsqn   number;
l_exit_status number;
l_skip_action number;
Begin
-- read all relevant info
dbms_logstdby_context.get_context(name => 'STATEMENT', value => l_stmt);
dbms_logstdby_context.get_context(name => 'PACKAGE_SCHEMA', value =>
l_pkgown);
dbms_logstdby_context.get_context(name => 'PACKAGE_NAME', value =>
l_pkgnam);
dbms_logstdby_context.get_context(name => 'PROCEDURE_NAME', value =>
l_procnm);
dbms_logstdby_context.get_context(name => 'CURRENT_SCHEMA', value =>
l_cur_schema);
dbms_logstdby_context.get_context(name => 'XIDUSN', value => l_xidusn);
dbms_logstdby_context.get_context(name => 'XIDSLT', value => l_xidslt);
dbms_logstdby_context.get_context(name => 'XIDSQN', value => l_xidsqn);
dbms_logstdby_context.get_context(name => 'EXIT_STATUS', value =>
l_ext_status);

if 0 == l_ext_status then
Insert Into sec_mgr.logit Values
      ('Success: '||l_pkgown||'.'||l_pkgnm||'.'||l_procnm|| ' by '||
l_current_user);

If l_current_user != 'TESTSCHEMA' Then
      l_skip_action := DBMS_LOGSTDBY.SKIP_ACTION_APPLY;
    Else
      l_skip_action := DBMS_LOGSTDBY.SKIP_ACTION_SKIP;
    End If;
  End If;

dbms_logstdby_context.set_context(name=>'SKIP_ACTION',
value => l_skip_action);

End skip_drop_policy;

EXECUTE DBMS_LOGSTDBY.SKIP( -
     stmt => 'PL/SQL', -
     schema_name => 'SYS', -
     object_name => 'DBMS_RLS.DROP_POLICY', -
     proc_name => 'SEC_MGR.SKIP_DROP_POLICY' -
     use_like=> FALSE);
```

Table 123-14 (Cont.) SKIP Procedure Parameters

Parameter	Description
use_like	Allows pattern matching to isolate the tables that you want to skip on the logical standby database. The use_like parameter matches a portion of one character value to another by searching the first value for the pattern specified by the second, and calculates strings using characters as defined by the input character set. This parameter follows the same rules for pattern matching described in *Oracle Database SQL Language Reference*.
esc	Identifies an escape character (such as the character "/") that you can use for pattern matching. If the escape character appears in the pattern before the character "%" or "_" then Oracle interprets this character literally in the pattern, rather than as a special pattern matching character. See *Oracle Database SQL Language Reference* for more information about pattern matching.

Usage Notes

- This procedure requires DBA privileges to execute.
- You cannot associate a stored procedure to be invoked in the context of a DML statement. For example, the following statement returns the ORA-16104: invalid Logical Standby option requested error:

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP(-
     stmt => 'DML', -
     schema_name => 'HR', -
     object_name => 'EMPLOYEES', -
     proc_name => 'DML_HANDLER');
```

  Also, if an event matches multiple rules either because of the use of wildcards while specifying the rule or because of a specification of overlapping rules. For example, if you specify a rule for the SCHEMA_DDL event for the HR.EMPLOYEES table, and a rule for the ALTER TABLE event for the HR.EMPLOYEES table, only one of the matching procedures will be invoked (alphabetically, by procedure). In the following code example, consider the following rules:

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP( -
     stmt => 'SCHEMA_DDL', -
     schema_name => 'HR', -
     object_name => 'EMPLOYEES', -
     proc_name => 'SCHEMA_DDL_HANDLER');
SQL> EXECUTE DBMS_LOGSTDBY.SKIP( -
     stmt => 'ALTER TABLE', -
     schema_name => 'HR', -
     object_name => 'EMPLOYEES', -
     proc_name => 'TABLE_ALTER_HANDLER');
```

  On encountering an ALTER TABLE statement, the schema_ddl_handler procedure will be invoked because its name will be at the top of an alphabetically sorted list of procedures that are relevant to the statement.Collisions on a rule set because of a specification containing wildcard entries are resolved in a similar fashion. For example, the rules in the following example will result in the empddl_handler procedure being invoked upon encountering the ALTER TABLE HR.EMPLOYEES ADD COLUMN RATING NUMBER statement:

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP(-
     stmt => 'ALTER TABLE', -
     schema_name => 'HR', -
     object_name => 'EMP%', -
```

```
     proc_name => 'EMPDDL_HANDLER');
SQL> EXECUTE DBMS_LOGSTDBY.SKIP( -
     stmt => 'ALTER TABLE', -
     schema_name => 'HR', -
     object_name => 'EMPLOYEES', -
     proc_name => 'EMPLOYEE_DDL_HANDLER');
```

- Use the SKIP procedure with caution, particularly when skipping DDL statements. If a CREATE TABLE statement is skipped, for example, you must also specify other DDL statements that refer to that table in the SKIP procedure. Otherwise, the statements will fail and cause an exception. When this happens, SQL Apply stops running.
- Before calling the SKIP procedure, SQL Apply must be halted. Do this by issuing an ALTER DATABASE STOP LOGICAL STANDBY APPLY statement. Once all desired filters have been specified, issue an ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE statement to start SQL Apply using the new filter settings.
- See the UNSKIP procedure for information about reversing (undoing) the settings of the SKIP procedure.
- For USER statements, the SCHEMA_NAME parameter will be the user and specify '%' for the OBJECT_NAME parameter.
- If the PROC_NAME parameter is supplied, it must already exist in DBA_PROCEDURES and it must execute with DEFINER rights. If the procedure is declared with INVOKER rights, the ORA-1031: insufficient privileges message will be returned.
- If the procedure returns a REPLACEMENT statement, the REPLACEMENT statement will be executed using the SYSTEM and OBJECT privileges of the owner of the procedure.
- The PL/SQL block of a SKIP procedure cannot contain transaction control statements (for example, COMMIT, ROLLBACK, SAVEPOINT, and SET CONSTRAINT) unless the block is declared to be an autonomous transaction.

Skip Statement Options

Table 123-15 lists the supported values for the stmt parameter of the SKIP procedure. The left column of the table lists the keywords that may be used to identify the set of SQL statements to the right of the keyword. In addition, any of the SQL statements listed in the sys.audit_actions table (shown in the right column of Table 123-15) are also valid values. Note that keywords are generally defined by database object.

Table 123-15 Supported Values for the stmt Parameter

Keyword	Associated SQL Statements
There is no keyword for this group of SQL statements.	GRANT REVOKE ANALYZE TABLE ANALYZE INDEX ANALYZE CLUSTER
CLUSTER	AUDIT CLUSTER CREATE CLUSTER DROP CLUSTER TRUNCATE CLUSTER
CONTAINER	See "Skipping Containers"

Table 123-15 (Cont.) Supported Values for the `stmt` Parameter

Keyword	Associated SQL Statements
CONTEXT	CREATE CONTEXT DROP CONTEXT
DATABASE LINK	CREATE DATABASE LINK CREATE PUBLIC DATABASE LINK DROP DATABASE LINK DROP PUBLIC DATABASE LINK
DIMENSION	ALTER DIMENSION CREATE DIMENSION DROP DIMENSION
DIRECTORY[1]	CREATE DIRECTORY DROP DIRECTORY
DML	Includes DML statements on a table (for example: INSERT, UPDATE, and DELETE)
INDEX	ALTER INDEX CREATE INDEX DROP INDEX
NON_SCHEMA_DDL	*All DDL that does not pertain to a particular schema* **Note:** SCHEMA_NAME and OBJECT_NAME must be null
PL/SQL[2]	Execute Oracle-supplied package.
PROCEDURE[3]	ALTER FUNCTION ALTER PACKAGE ALTER PACKAGE BODY ALTER PROCEDURE CREATE FUNCTION CREATE LIBRARY CREATE PACKAGE CREATE PACKAGE BODY CREATE PROCEDURE DROP FUNCTION DROP LIBRARY DROP PACKAGE DROP PACKAGE BODY DROP PROCEDURE
PROFILE	ALTER PROFILE CREATE PROFILE DROP PROFILE

Table 123-15 (Cont.) Supported Values for the `stmt` Parameter

Keyword	Associated SQL Statements
ROLE	ALTER ROLE CREATE ROLE DROP ROLE SET ROLE
ROLLBACK STATEMENT	ALTER ROLLBACK SEGMENT CREATE ROLLBACK SEGMENT DROP ROLLBACK SEGMENT
SCHEMA_DDL	*All DDL statements that create, modify, or drop schema objects (for example: tables, indexes, and columns)* **Note:** SCHEMA_NAME and OBJECT_NAME must *not* be null
SEQUENCE	ALTER SEQUENCE CREATE SEQUENCE DROP SEQUENCE
SYNONYM	CREATE PUBLIC SYNONYM CREATE SYNONYM DROP PUBLIC SYNONYM DROP SYNONYM
SYSTEM AUDIT	AUDIT *SQL_statements* NOAUDIT *SQL_statements*
TABLE	CREATE TABLE ALTER TABLE DROP TABLE TRUNCATE TABLE
TABLESPACE	CREATE TABLESPACE DROP TABLESPACE ALTER TABLESPACE
TRIGGER	ALTER TRIGGER CREATE TRIGGER DISABLE ALL TRIGGERS DISABLE TRIGGER DROP TRIGGER ENABLE ALL TRIGGERS ENABLE TRIGGER

Table 123-15 (Cont.) Supported Values for the `stmt` Parameter

Keyword	Associated SQL Statements
TYPE	ALTER TYPE ALTER TYPE BODY CREATE TYPE CREATE TYPE BODY DROP TYPE DROP TYPE BODY
USER	ALTER USER CREATE USER DROP USER
VIEW	CREATE VIEW DROP VIEW
VIEW	CREATE VIEW DROP VIEW

[1] All directory objects are owned by SYS, but for the purpose of filtering them with a skip directive the schema should be specified as '%'".
[2] See *Oracle Data Guard Concepts and Administration* for information about supported packages.
[3] Java schema objects (sources, classes, and resources) are considered the same as procedure for purposes of skipping (ignoring) SQL statements.

Exceptions

Table 123-16 DBMS_LOGSTDBY.SKIP Procedure Exceptions

Exception	Description
ORA-01031	Insufficient privileges: • Procedure used INVOKER rights • Procedure needs DBA privileges
ORA-16103	Logical standby apply must be stopped to allow this operation.
ORA-16104	Invalid logical standby option requested.
ORA-16203	"Unable to interpret SKIP procedure return values." Indicates that a SKIP procedure has either generated an exception or has returned ambiguous values. You can identify the offending procedure by examining the DBA_LOGSTDBY_EVENTS view.
ORA-16236	Logical standby metadata operation in progress.

Examples

Example 1. Skipping all DML and DDL changes made to a schema
The following example shows how to specify rules so that SQL Apply will skip both DDL and DML statements made to the HR schema.

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP(STMT => 'SCHEMA_DDL', -
     schema_name => 'HR', -
     object_name => '%', -
     proc_name => null);
SQL> EXECUTE DBMS_LOGSTDBY.SKIP(STMT => 'DML', -
     schema_name => 'HR', -
     object_name => '%', -
     proc_name => null);
```

Example 2. Creating a procedure to handle different file system organization
For example, if the file system organization in the logical standby database is different than that in the primary database, you can write a `SKIP` procedure to handle DDL statements with file specifications transparently.The following procedure can handle DDL statements as long as you follow a specific naming convention for the file specification string.

1. Create the `SKIP` procedure to handle tablespace DDL statements:

```
CREATE OR REPLACE PROCEDURE sys.handle_tbs_ddl
IS
l_old_stmt varchar2(4000);
l_stmt_typ varcahr2(40);
l_schema   varchar2(30);
l_name     varchar2(30);
l_xidusn   number;
l_xidslt   number;
l_xidsqn   number;
l_skip_action number;
l_new_stmt varchar2(4000);

-- read all information
dbms_logstdby_context.get_context(name=>'STATEMENT',value=>l_old_stmt);
  dbms_logstdby_context.get_context(name=>'STATEMENT_TYPE',value=>l_stmt_type);
  dbms_logstdby_context.get_context(name=>'OWNER',value=>l_schema);
  dbms_logstdby_context.get_context(name=>'NAME',value=>l_name);
  dbms_logstdby_context.get_context(name=>'XIDUSN',value=>l_xidusn);
  dbms_logstdby_context.get_context(name=>'XIDSLT',value=>l_xidslt);
  dbms_logstdby_context.get_context(name=>'XIDSQN',value=>l_xidsqn);
  dbms_logstdby_context.get_context(name=>'CONTAINER_NAME',value=>l_conname);

--
-- All primary file specification that contains a directory
-- /usr/orcl/primary/dbs
-- should go to /usr/orcl/stdby directory specification

  BEGIN
  l_new_stmt := replace (l_old_stmt, '/usr/orcl/primary/dbs','/usr/orcl/stdby');
l_skip_action := DBMS_LOGSTDBY.SKIP_ACTION_REPLACE;
  EXCEPTION
  WHEN OTHERS THEN
    l_skip_action := DBMS_LOGSTDBY.SKIP_ACTION_ERROR;
    l_new_stmt := NULL;
  END;

  dbms_logstdby_context.set_context(name=>new_statement, value => l_new_stmt);
dbms_logstdby_context.set_context(name=>'SKIP_ACTION', value => l_skip_action);
END handle_tbs_ddl;
```

2. Register the `SKIP` procedure with SQL Apply:

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP (stmt => 'TABLESPACE', -
             proc_name => 'SYS.HANDLE_TBS_DDL');
```

Skipping Containers

To skip a container (either a PDB or the root), use the `CONTAINER` keyword. All SQL statements executed on the container, as well as any other actions taken on the container, are skipped.

You can skip a particular PDB within a CDB. For example, the following command skips the PDB named `PDB1`. The command must be executed at the root level:

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP(stmt => 'CONTAINER', object_name => 'PDB1');
```

As shown in the following example, you could also skip only the root of the CDB, but not any of the PDBs that exist under the root. The command must be executed at the root level:

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP(stmt => 'CONTAINER', object_name => 'CDB$ROOT');
```

> **Note:**
>
> To create other skip rules for a container, create the rules from within the container. The container to which the rules will apply is automatically derived from the container in which the rules are created.

SKIP_ERROR Procedure

The SKIP_ERROR procedure specifies the action to take when a logical standby database detects an error.

Upon encountering an error, the logical standby database uses the criteria contained in this procedure to determine a course of action. The default action when a match is found is to skip the error and continue with applying changes. However, if a procedure is supplied, then `SKIP_ERROR` can take other actions depending on the situation. It can do nothing, which causes SQL Apply to stop, or it can change the error message text and stop SQL Apply, or it can actually skip the error.

Syntax

```
DBMS_LOGSTDBY.SKIP_ERROR (
     stmt                      IN VARCHAR2,
     schema_name               IN VARCHAR2 DEFAULT NULL,
     object_name               IN VARCHAR2 DEFAULT NULL,
     proc_name                 IN VARCHAR2 DEFAULT NULL,
     use_like                  IN BOOLEAN  DEFAULT NULL,
     esc                       IN CHAR1    DEFAULT NULL);
```

Parameters

Table 123-17 SKIP_ERROR Procedure Parameters

Parameter	Description
`stmt`	Either a keyword that identifies a set of SQL statements or a specific SQL statement. The use of keywords simplifies configuration because keywords, generally defined by the database object, identify all SQL statements that operate on the specified object. Table 123-15 shows a list of keywords and the equivalent SQL statements, either of which is a valid value for this parameter.

Table 123-17 (Cont.) SKIP_ERROR Procedure Parameters

Parameter	Description
schema_name	The name of one or more schemas (wildcards are permitted) associated with the SQL statements identified by the stmt parameter. If not applicable, this value must be set to NULL.
object_name	The name of one or more objects (wildcards are permitted) associated with the SQL statements identified by the stmt. If not applicable, this value must be set to NULL.
proc_name	Name of a stored procedure to call when SQL Apply encounters an error and determines a particular statement matches the filter defined by the stmt, schema_name, and object_name parameters. Specify the procedure in the following format: '"schema"."package"."procedure"' This procedure returns an error message that directs SQL Apply to perform one of the following actions: • Silently skip the error and continue with SQL Apply • Replace the error message that would have been created with a custom one, and stop SQL Apply • Do nothing, causing SQL Apply to stop and the original error message to be logged The procedure registered with SQL Apply does not take any parameters. The context associated with LSBY_APPLY_CONTEXT can be used to retrieve all relevant information related to the error. See the DBMS_LOGSTDBY_CONTEXT package for a list of all parameters associated with LSBY_APPLY_CONTEXT. The parameters of interest for procedures registered with SKIP_ERROR are CONTAINER_NAME, STATEMENT, STATEMENT_TYPE, SCHEMA, NAME, XIDUSN, XIDSLT, XIDSQN, ERROR and NEW_ERROR.
use_like	Allows pattern matching to isolate the tables that you want to skip on the logical standby database. The use_like parameter matches a portion of one character value to another by searching the first value for the pattern specified by the second, and calculates strings using characters as defined by the input character set. This parameter follows the same rules for pattern matching described in *Oracle Database SQL Language Reference*.
esc	Identifies an escape character (such as the characters "%" or "_") that you can use for pattern matching. If the escape character appears in the pattern before the character "%" or "_" then Oracle interprets this character literally in the pattern, rather than as a special pattern matching character.

Usage Notes

- A stored procedure provided to the SKIP_ERROR procedure is called when SQL Apply encounters an error that could shut down the application of redo logs to the standby database.
- Running this stored procedure affects the error being written in the STATUS column of the DBA_LOGSTDBY_EVENTS table. The STATUS_CODE column remains unchanged. If the stored procedure is to have no effect, that is, apply will be stopped, then the NEW_ERROR is written to the events table. To truly have no effect, set NEW_ERROR to ERROR in the procedure.
- If the stored procedure requires that a shutdown be avoided, then you must set NEW_ERROR to NULL.
- This procedure requires DBA privileges to execute.

- For USER statements, the SCHEMA_NAME parameter will be the user and you should specify '%' for the OBJECT_NAME parameter.
- If the PROC_NAME parameter is specified, it must already exist in DBA_PROCEDURES and it must execute with DEFINERS rights. If the procedure is declared with INVOKERS rights, the ORA-1031: insufficient privileges message will be returned.
- The PL/SQL block of a SKIP_ERROR procedure cannot contain transaction control statements (for example: COMMIT, ROLLBACK, SAVEPOINT, and SET CONSTRAINT) unless the block is declared to be an autonomous transaction using the following syntax:

```
PRAGMA AUTONOMOUS_TRANSACTION
```

Exceptions

Table 123-18 SKIP_ERROR Procedure Exceptions

Exception	Description
ORA-01031	Insufficient privileges: • Procedure used INVOKER rights • Procedure needs DBA privileges
ORA-16103	Logical Standby apply must be stopped to allow this operation
ORA-16104	invalid Logical Standby option requested
ORA-16236	Logical Standby metadata operation in progress

Example 1

The following example shows how to specify rules so that SQL Apply will skip any error raised from any GRANT DDL command.

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP_ERROR('GRANT')
```

Example 2

To skip errors on GRANT statements on SYS or HR schemas, define a procedure handle_error_ddl and register it. In the following example, assume that handle_error_ddl is a free-standing procedure in the SYS schema.

1. Create the error-handler procedure:

```
CREATE OR REPLACE PROCEDURE sys.handle_error_ddl
is
l_stmt    VARCHAR2(4000);
l_stmt_type VARCHAR2(40);
l_schema    VARCHAR2(30);
l_name      VARCHAR2(30);
l_xidusn      NUMBER;
l_xidslt      NUMBER;
l_xidsqn      NUMBER;
l_error     VARCHAR2(4000);
l_conname    VARCHAR2(30);
l_newerr   VARCHAR2(4000);

BEGIN
 dbms_logstdby_context.get_context(name=>'STATEMENT',value=>l_stmt);
  dbms_logstdby_context.get_context(name=>'STATEMENT_TYPE',value=>l_stmt_type);
  dbms_logstdby_context.get_context(name=>'SCHEMA',value=>l_schema);
  dbms_logstdby_context.get_context(name=>'NAME',value=>l_name);
```

```
  dbms_logstdby_context.get_context(name=>'XIDUSN',value=>l_xidusn);
  dbms_logstdby_context.get_context(name=>'XIDSLT',value=>l_xidslt);
  dbms_logstdby_context.get_context(name=>'XIDSQN',value=>l_xidsqn);
  dbms_logstdby_context.get_context(name=>'ERROR',value=>l_error);
  dbms_logstdby_context.get_context(name=>'CONTAINER_NAME',value=>l_conname);

-- default error to what we already have
l_new_error := l_error;

-- Ignore any GRANT errors on SYS or HR schemas

IF INSTR(UPPER(l_stmt), 'GRANT') > 0
THEN
IF l_schema is NULL
OR (l_schema is NOT NULL AND
(UPPER(l_schema) = 'SYS' OR
UPPER(l_schema) = 'HR')
THEN
l_new_error := NULL;
-- record the fact that we just skipped an error on 'SYS' or 'HR' schemas
-- code not shown here
END IF;
END IF;

dbms_logstdby_context.set_context(name => 'NEW_ERROR', value => l_new_error);

END handle_error_ddl;
/
```

2. Register the error handler with SQL Apply:

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP_ERROR ( -
     statement => 'NON_SCHEMA_DDL', -
     schema_name => NULL, -
     object_name => NULL, -
     proc_name => 'SYS.HANDLE_ERROR_DDL');
```

SKIP_TRANSACTION Procedure

This procedure provides a way to skip (ignore) applying transactions to the logical standby database. You can skip specific transactions by specifying transaction identification information.

Syntax

```
DBMS_LOGSTDBY.SKIP_TRANSACTION (
     xidusn_p          IN NUMBER,
     xidslt_p          IN NUMBER,
     xidsqn_p          IN NUMBER,
     con_name_p      VARCHAR2     IN     DEFAULT  NULL
);
```

Parameters

Table 123-19 SKIP_TRANSACTION Procedure Parameters

Parameter	Description
xidusn_p NUMBER	Transaction ID undo segment number of the transaction being skipped
xidslt_p NUMBER	Transaction ID slot number of the transaction being skipped

Table 123-19 (Cont.) SKIP_TRANSACTION Procedure Parameters

Parameter	Description
xidsqn_p NUMBER	Transaction ID sequence number of the transaction being skipped
con_name_p	The transaction name.

Usage Notes

If SQL Apply stops due to a particular transaction (for example, a DDL transaction), you can specify that transaction ID and then continue to apply. You can call this procedure multiple times for as many transactions as you want SQL Apply to ignore.

> **WARNING:**
>
> **SKIP_TRANSACTION** is an inherently dangerous operation. Do not invoke this procedure unless you have examined the transaction in question through the **V$LOGMNR_CONTENTS** view and have taken compensating actions at the logical standby database. **SKIP_TRANSACTION** is not the appropriate procedure to invoke to skip DML changes to a table.
>
> To skip a DML failure, use a **SKIP** procedure, such as **SKIP('DML','MySchema','MyFailed Table').** Using the **SKIP_TRANSACTION** procedure for DML transactions may skip changes for other tables, thus logically corrupting them.

- This procedure requires DBA privileges to execute.
- Use the DBA_LOGSTDBY_SKIP_TRANSACTION view to list the transactions that are going to be skipped by SQL Apply.

Exceptions

Table 123-20 SKIP_TRANSACTION Procedure Exceptions

Exception	Description
ORA-01031	Need DBA privileges
ORA-16103	Logical Standby apply must be stopped to allow this operation
ORA-16104	invalid Logical Standby option requested

Examples

To skip a DDL transaction with (xidusn_p, xidslt_p, xidsqn_p) of (1.13.1726) you can register a rule as shown in the following example:

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP_TRANSACTION (-
     xidusn_p => 1, xidslt_p => 13, xidsqn_p => 1726);
```

UNSKIP Procedure

Use the UNSKIP procedure to delete rules specified earlier with the SKIP procedure.

The parameters specified in the UNSKIP procedure must match exactly for it to delete an already-specified rule.

The container_name argument is valid only in a CDB.

Syntax

```
DBMS_LOGSTDBY.UNSKIP (
     stmt                      IN VARCHAR2,
     schema_name               IN VARCHAR2 DEFAULT NULL,
     object_name               IN VARCHAR2 DEFUALT NULL,
     container_name            IN VARCHAR2 DEFAULT NULL);
```

Parameters

The parameter information for the UNSKIP procedure is the same as that described for the SKIP procedure. See Table 123-14 for complete parameter information.

Exceptions

Table 123-21 UNSKIP Procedure Exceptions

Exception	Description
ORA-01031	need DBA privileges to execute this procedure
ORA-16103	Logical Standby apply must be stopped to allow this operation
ORA-16104	invalid Logical Standby option requested

Usage Notes

> **WARNING:**
>
> If DML changes for a table have been skipped and not compensated for, you must follow the call to the **UNSKIP** procedure with a call to the **INSTANTIATE_TABLE** procedure to synchronize this table with those maintained by SQL Apply.

- This procedure requires DBA privileges to execute.
- Wildcards passed in the schema_name or the object_name parameter are not expanded. The wildcard character is matched at the character level. Thus, you can delete only one specified rule by invoking the UNSKIP procedure, and you will need a distinct UNSKIP procedure call to delete each rule that was previously specified.

  For example, assume you have specified the following two rules to skip applying DML statements to the HR.EMPLOYEE and HR.EMPTEMP tables:

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP (STMT => 'DML',-
     SCHEMA_NAME => 'HR', -
     OBJECT_NAME => 'EMPLOYEE', -
     PROC_NAME => null);
```

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP (STMT => 'DML',-
     SCHEMA_NAME => 'HR', -
     OBJECT_NAME => 'EMPTEMP', -
     PROC_NAME => null);
```

In the following example, the wildcard in the TABLE_NAME parameter cannot be used to delete the rules that were specified:

```
SQL> EXECUTE DBMS_LOGSTDBY.UNSKIP (STMT => 'DML',-
     SCHEMA_NAME => 'HR', -
     OBJECT_NAME => 'EMP%');
```

In fact, this UNSKIP procedure matches neither of the rules, because the wildcard character in the TABLE_NAME parameter is not expanded. Instead, the wildcard character will be used in an exact match to find the corresponding SKIP rule.

UNSKIP_ERROR Procedure

Use the UNSKIP_ERROR procedure to delete rules specified earlier with the SKIP_ERROR procedure.

The parameters specified in the UNSKIP_ERROR procedure must match exactly for the procedure to delete an already-specified rule.

The container_name argument is valid only in a CDB.

Syntax

```
DBMS_LOGSTDBY.UNSKIP_ERROR (
     stmt                      IN VARCHAR2,
     schema_name               IN VARCHAR2 DEFAULT NULL,
     object_name               IN VARCHAR2 DEFAULT NULL,
     container_name            IN VARCHAR2 DEFAULT NULL);
```

Parameters

The parameter information for the UNSKIP_ERROR procedure is the same as that described for the SKIP_ERROR procedure. See Table 123-17 for complete parameter information.

Exceptions

Table 123-22 UNSKIP_ERROR Procedure Exceptions

Exception	Description
ORA-01031	Need DBA privileges
ORA-16103	Logical Standby apply must be stopped to allow this operation
ORA-16104	invalid Logical Standby option requested

Usage Notes

- This procedure requires DBA privileges to execute.
- Wildcards passed in the schema_name or the object_name parameters are not expanded. Instead, the wildcard character is treated as any other character and an exact match is made. Thus, you can delete only one specified rule by invoking the UNSKIP_ERROR procedure, and you need a distinct UNSKIP_ERROR procedure call to delete each rule that you previously specified.

For example, assume you have specified the following two rules to handle the HR.EMPLOYEE and HR.EMPTEMP tables:

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP_ERROR (STMT => 'DML',-
     SCHEMA_NAME => 'HR', -
     OBJECT_NAME => 'EMPLOYEE', -
     PROC_NAME => 'hr_employee_handler');
SQL> EXECUTE DBMS_LOGSTDBY.SKIP_ERROR (STMT => 'DML',-
     SCHEMA_NAME => 'HR', -
     OBJECT_NAME => 'EMPTEMP', -
     PROC_NAME => 'hr_tempemp_handler');
```

In this case, the following UNSKIP procedure cannot be used to delete the rules that you have specified:

```
SQL> EXECUTE DBMS_LOGSTDBY.UNSKIP_ERROR (STMT => 'DML',-
     SCHEMA_NAME => 'HR', -
     OBJECT_NAME => 'EMP%');
```

In fact, the UNSKIP procedure will match neither of the rules, because the wildcard character in the OBJECT_NAME parameter will not be expanded.

Example

To remove a handler that was previously registered with SQL Apply from getting called on encountering an error, you can issue the following statement:

```
DBMS_LOGSTDBY.UNSKIP_ERROR ( -
      statement => 'NON_SCHEMA_DDL', -
      schema_name => NULL, -
      object_name => NULL);
```

UNSKIP_TRANSACTION Procedure

Use the UNSKIP_TRANSACTION procedure to delete rules specified earlier with the SKIP_TRANSACTION procedure.

The parameters specified in the UNSKIP_TRANSACTION procedure must match exactly for the procedure to delete an already-specified rule.

Syntax

```
DBMS_LOGSTDBY.UNSKIP_TRANSACTION (
     xidusn_p         IN NUMBER,
     xidslt_p         IN NUMBER,
     xidsqn_p         IN NUMBER);
```

Parameters

Table 123-23 UNSKIP_TRANSACTION Procedure Parameters

Parameter	Description
XIDUSN	Transaction ID undo segment number of the transaction being skipped
XIDSLT	Transaction ID slot number of the transaction being skipped
XIDSQN	Transaction ID sequence number of the transaction being skipped

Exceptions

Table 123-24 UNSKIP_TRANSACTION Procedure Exceptions

Exception	Description
ORA-01031	need DBA privileges to execute this procedure
ORA-16103	Logical Standby apply must be stopped to allow this operation
ORA-16104	invalid Logical Standby option requested

Usage Notes

- This procedure requires DBA privileges to execute.
- Query the DBA_LOGSTDBY_SKIP_TRANSACTION view to list the transactions that are going to be skipped by SQL Apply.

Examples

To remove a rule that was originally specified to skip the application of a transaction with (XIDUSN, XIDSLT, XIDSQN) of (1.13.1726) issue the following statement:

```
SQL> DBMS_LOGSTDBY.UNSKIP_TRANSACTION (XIDUSN => 1, XIDSLT => 13, XIDSQN => 1726);
```