Oracle® Streams

Concepts and Administration

10*g* Release 2 (10.2)

B14229-01

June 2005

ORACLE®

Oracle Streams Concepts and Administration, 10*g* Release 2 (10.2)

B14229-01

Primary Author: Randy Urbano

Contributors: Sundeep Abraham, Nimar Arora, Lance Ashdown, Ram Avudaiappan, Sukanya Balaraman, Neerja Bhatt, Ragamayi Bhyravabhotla, Chipper Brown, Diego Cassinera, Debu Chatterjee, Jack Chung, Alan Downing, Lisa Eldridge, Curt Elsbernd, Yong Feng, Diana Foch, Jairaj Galagali, Brajesh Goyal, Connie Green, Sanjay Kaluskar, Ravikanth Kasamsetty, Lewis Kaplan, Joydip Kundu, Anand Lakshminath, Jing Liu, Edwina Lu, Raghu Mani, Pat McElroy, Krishnan Meiyyappan, Shailendra Mishra, Bhagat Nainani, Anand Padmanaban, Maria Pratt, Arvind Rajaram, Viv Schupmann, Vipul Shah, Neeraj Shodhan, Wayne Smith, Benny Souder, Jim Stamos, Janet Stern, Mahesh Subramaniam, Kapil Surlaker, Bob Thome, Hung Tran, Ramkumar Venkatesan, Byron Wang, Wei Wang, James M. Wilson, Lik Wong, David Zhang

Contents

2 Streams Capture Process

3 Streams Staging and Propagation

4 Streams Apply Process

5 Rules

6 How Rules Are Used in Streams

7 Rule-Based Transformations

8 Information Provisioning

9 Streams High Availability Environments

12 Managing Staging and Propagation

13 Managing an Apply Process

14 Managing Rules

21 Monitoring Streams Queues and Propagations

22 Monitoring Streams Apply Processes

23 Monitoring Rules

24 Monitoring Rule-Based Transformations

25 Monitoring File Group and Tablespace Repositories

26 Monitoring Other Streams Components

C Online Database Maintenance with Streams

Preface

Oracle Streams Concepts and Administration describes the features and functionality of Streams. This document contains conceptual information about Streams, along with information about managing a Streams environment. In addition, this document contains detailed examples that configure a Streams capture and apply environment and a rule-based application.

This Preface contains these topics:

Audience

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

- Plan for a Streams environment
- Configure a Streams environment
- Administer a Streams environment
- Monitor a Streams environment
- Perform necessary troubleshooting activities

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

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at

`http://www.oracle.com/accessibility/`

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

Related Documents

For more information, see these Oracle resources:

- *Oracle Streams Replication Administrator's Guide*
- *Oracle Database Concepts*
- *Oracle Database Administrator's Guide*
- *Oracle Database SQL Reference*
- *Oracle Database PL/SQL Packages and Types Reference*
- *Oracle Database PL/SQL User's Guide and Reference*
- *Oracle Database Utilities*
- *Oracle Database Heterogeneous Connectivity Administrator's Guide*
- *Oracle Streams Advanced Queuing User's Guide and Reference*
- Streams online help for the Streams tool in Oracle Enterprise Manager

Many of the examples in this book use the sample schemas of the sample database, which is installed by default when you install Oracle Database. Refer to *Oracle Database Sample Schemas* for information on how these schemas were created and how you can use them yourself.

Printed documentation is available for sale in the Oracle Store at

http://oraclestore.oracle.com/

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at

http://www.oracle.com/technology/membership/

If you already have a username and password for OTN, then you can go directly to the documentation section of the OTN Web site at

http://www.oracle.com/technology/documentation/

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

What's New in Oracle Streams?

This section describes new features of Oracle Streams for Oracle Database 10*g* Release 2 (10.2) and provides pointers to additional information. New features information from previous releases is also retained to help those users migrating to the current release.

The following sections describe the new features in Oracle Streams:

- Oracle Database 10g Release 2 (10.2) New Features in Streams
- Oracle Database 10g Release 1 (10.1) New Features in Streams

Oracle Database 10*g* Release 2 (10.2) New Features in Streams

The following sections describe the new features in Oracle Streams for Oracle Database 10*g* Release 2 (10.2):

- Streams Performance Improvements
- Streams Configuration and Manageability Enhancements
- Streams Replication Enhancements
- Rules Interface Enhancement
- Information Provisioning Enhancements

Streams Performance Improvements

Oracle Database 10*g* Release 2 includes performance improvements for most Streams operations. Specifically, the following Streams components have been improved to perform more efficiently and handle greater workloads:

- Capture processes
- Propagations
- Apply processes

This release also includes the following specific performance improvements:

- More types of **rule**s are simple rules for faster rule evaluation. See "Simple Rule Conditions" on page 5-3.
- Declarative rule-based transformations perform transformations more efficiently. See "Declarative Rule-Based Transformations" on page 7-1.
- Real-time downstream capture reduces the amount of time required for a **downstream capture process** to capture changes made at the **source database**. See "Real-Time Downstream Capture" on page 2-14.

- Enhanced prefiltering during **capture process** rule evaluation enables capture processes to capture changes in the redo log more efficiently. See "Capture Process Rule Evaluation" on page 2-41.
- The new `ANYDATA_FAST_EVAL_FUNCTION` function in the `STREAMS$_EVALUATION_CONTEXT` provides more efficient access to values inside an `ANYDATA` object. See "Evaluation Contexts Used in Streams" on page 6-34.

Streams Configuration and Manageability Enhancements

The following are Streams configuration manageability enhancements for Oracle Database 10*g* Release 2:

- Automatic Shared Memory Management of the Streams Pool
- Streams Tool in Oracle Enterprise Manager
- Procedures for Starting and Stopping Propagations
- Queue-to-Queue Propagations
- Declarative Rule-Based Transformations
- Commit-Time Queues
- Supplemental Logging Enabled During Preparation for Instantiation
- Configurable Transaction Spill Threshold for Apply Processes
- Conversion of LCRs to and from XML
- Retrying an Error Transaction with a User Procedure
- Enhanced Support for Index-Organized Tables
- Row LCR Execution Enhancements
- Information About Oldest Transaction in V$STREAMS_APPLY_READER

Automatic Shared Memory Management of the Streams Pool

The Oracle Automatic Shared Memory Management feature manages the size of the **Streams pool** when the `SGA_TARGET` initialization parameter is set to a nonzero value.

> **See Also:** "Streams Pool" on page 3-19

Streams Tool in Oracle Enterprise Manager

The Streams tool in Oracle Enterprise Manager enables you to configure, manage, and monitor a Streams environment using a Web browser.

> **See Also:**
>
> - "Streams Tool in the Oracle Enterprise Manager Console" on page 1-20
> - The online help for the Streams tool in Oracle Enterprise Manager

Procedures for Starting and Stopping Propagations

The `START_PROPAGATION` and `STOP_PROPAGATION` procedures are added to the `DBMS_PROPAGATION_ADM` package.

See Also:

- "Starting a Propagation" on page 12-9
- "Stopping a Propagation" on page 12-10

Queue-to-Queue Propagations

A queue-to-queue **propagation** always has its own exclusive **propagation job** to propagate **message**s from the **source queue** to the **destination queue**. Also, in an Oracle Real Application Clusters (RAC) environment, when the destination queue in a queue-to-queue propagation is a **buffered queue**, the queue-to-queue propagation uses a service for transparent failover to another instance if the primary RAC instance fails.

See Also: "Queue-to-Queue Propagations" on page 3-5

Declarative Rule-Based Transformations

Declarative rule-based transformations provide a simple interface for configuring a set of common transformation scenarios for row LCRs. No user-defined PL/SQL function is required to configure a **declarative rule-based transformation**.

See Also: "Declarative Rule-Based Transformations" on page 7-1

Commit-Time Queues

Commit-time queues provide more control over the order in which **user-enqueued message**s in a **queue** are browsed or dequeued.

See Also: "Commit-Time Queues" on page 3-14

Supplemental Logging Enabled During Preparation for Instantiation

The following procedures in the `DBMS_CAPTURE_ADM` package now include a `supplemental_logging` parameter which controls the **supplemental logging** specifications for the database objects being prepared for **instantiation**: `PREPARE_TABLE_INSTANTIATION`, `PREPARE_SCHEMA_INSTANTIATION`, and `PREPARE_GLOBAL_INSTANTIATION`.

See Also: *Oracle Streams Replication Administrator's Guide*

Configurable Transaction Spill Threshold for Apply Processes

The new `txn_lcr_spill_threshold` apply process parameter enables you to specify that an apply process begins to spill **message**s for a transaction from memory to disk when the number of messages in memory for a particular transaction exceeds the specified number. The `DBA_APPLY_SPILL_TXN` and `V$STREAMS_APPLY_READER` views enable you to monitor the number of transactions and messages spilled by an apply process.

See Also: *Oracle Database PL/SQL Packages and Types Reference*

Conversion of LCRs to and from XML

The following functions in the DBMS_STREAMS package convert a **logical change record (LCR)** to or from XML:

- CONVERT_LCR_TO_XML converts an LCR encapsulated in a ANYDATA object into an XML object that conforms to the XML schema for LCRs.
- CONVERT_XML_TO_LCR converts an XML object that conforms to the XML schema for LCRs into an LCR encapsulated in a ANYDATA object.

> **See Also:** *Oracle Database PL/SQL Packages and Types Reference*

Retrying an Error Transaction with a User Procedure

A new parameter, user_procedure, is added to the EXECUTE_ERROR procedure in the DBMS_APPLY_ADM package. This parameter enables you to specify a user procedure that modifies one or more LCRs in an error transaction before the transaction is executed.

> **See Also:** "Retrying a Specific Apply Error Transaction with a User Procedure" on page 13-24

Enhanced Support for Index-Organized Tables

Streams **capture process**es and **apply process**es now support index-organized tables that contain the following datatypes, in addition to the datatypes that were supported in past releases of Oracle:

- LONG
- LONG RAW
- CLOB
- NCLOB
- BLOB
- BFILE

Logical change records (LCRs) containing these datatypes in index-organized tables can also be propagated using **propagation**s.

Also, Streams now supports index-organized tables that include an OVERFLOW segment.

Row LCR Execution Enhancements

In previous releases, the EXECUTE member procedure for row LCRs only execute row LCRs in an apply handler for an apply process. In Oracle Database 10*g* Release 2, the EXECUTE member procedure can execute user-constructed row LCRs, row LCRs in the error queue, and row LCRs that were last enqueued by an apply process, user, or application.

> **See Also:**
>
> - *Oracle Database PL/SQL Packages and Types Reference*
> - *Oracle Streams Replication Administrator's Guide*

Information About Oldest Transaction in V$STREAMS_APPLY_READER

The following new columns are added to the V$STREAMS_APPLY_READER dynamic performance view: OLDEST_XIDUSN, OLDEST_XIDSLT, and OLDEST_XIDSQN. These columns show the transaction identification number of the oldest transaction being assembled or applied by an apply process. The DBA_APPLY_PROGRESS view also contains this information. However, for a running apply process, the information in the V$STREAMS_APPLY_READER view is more current than the information in the DBA_APPLY_PROGRESS view.

> **See Also:** *Oracle Database Reference* for more information about the V$STREAMS_APPLY_READER dynamic performance view

Streams Replication Enhancements

The following are Streams **replication** enhancements for Oracle Database 10*g* Release 2:

- Simple Streams Replication Configuration
- LOB Assembly
- Virtual Dependency Definitions
- Instantiation Using Transportable Tablespace from Backup
- RMAN Database Instantiation Across Platforms
- Apply Processes Allow Duplicate Rows
- View for Monitoring Long Running Transactions

Simple Streams Replication Configuration

The following new procedures in the DBMS_STREAMS_ADM package provide simplify configuration of a Streams **replication** environment:

- MAINTAIN_GLOBAL configures a Streams environment that replicates changes at the database level between two databases.
- MAINTAIN_SCHEMAS configures a Streams environment that replicates changes to specified schemas between two databases.
- MAINTAIN_SIMPLE_TTS configures a Streams environment that replicates changes to a single, self-contained tablespace between two databases. This procedure replaces the MAINTAIN_SIMPLE_TABLESPACE procedure.
- MAINTAIN_TABLES configures a Streams environment that replicates changes to specified tables between two databases.
- MAINTAIN_TTS configures a Streams environment that replicates changes to a self-contained set of tablespaces. This procedure replaces the MAINTAIN_TABLESPACES procedure.
- PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP configure a Streams environment that replicates changes at the database level or to specified tablespaces between two databases. These procedures must be used together, and **instantiation** actions must be performed manually, to complete the Streams replication configuration.

> **See Also:**
>
> - *Oracle Streams Replication Administrator's Guide*
> - *Oracle Database PL/SQL Packages and Types Reference*

LOB Assembly

LOB assembly simplifies processing of row LCRs with LOB columns in **DML handler** and **error handler**s.

> **See Also:** *Oracle Streams Replication Administrator's Guide*

Virtual Dependency Definitions

A virtual dependency definition is a description of a dependency that is used by an **apply process** to detect dependencies between transactions at a **destination database**. Virtual dependency definitions enable an apply process to detect dependencies that it would not be able to detect by using only the constraint information in the data dictionary.

> **See Also:** *Oracle Streams Replication Administrator's Guide*

Instantiation Using Transportable Tablespace from Backup

A new RMAN command, `TRANSPORT TABLESPACE`, enables you to instantiate a set of tablespaces while the tablespaces in the **source database** remain online. The tablespaces can be added to the **destination database** using Data Pump import or the `ATTACH_TABLESPACES` procedure in the `DBMS_STREAMS_TABLESPACE_ADM` package.

> **See Also:** *Oracle Streams Replication Administrator's Guide*

RMAN Database Instantiation Across Platforms

The RMAN `CONVERT DATABASE` command can be used to instantiate an entire database in a **replication** environment where the source and **destination database**s are running on different platforms that have the same endian format.

> **See Also:** *Oracle Streams Replication Administrator's Guide*

Apply Processes Allow Duplicate Rows

In releases prior to Oracle Database 10*g* Release 2, an apply process always raises an error when it encounters a row LCR that changes more than one row in a table. In Oracle Database 10*g* Release 2, the new `allow_duplicate_rows` apply process parameter can be set to `true` to allow an apply process to apply a row LCR that changes more than one row.

> **See Also:** *Oracle Database PL/SQL Packages and Types Reference*

View for Monitoring Long Running Transactions

The `V$STREAMS_TRANSACTION` dynamic performance view enables monitoring of long running transactions that currently are being processes by Streams **capture process**es and **apply process**es.

> **See Also:** *Oracle Database Reference* for more information about the `V$STREAMS_TRANSACTION` dynamic performance view

Rules Interface Enhancement

In Oracle Database 10*g* Release 2, a new procedure, `ALTER_EVALUATION_CONTEXT` in the `DBMS_RULE_ADM` package, enables you to alter an existing **evaluation context**.

See Also: *Oracle Database PL/SQL Packages and Types Reference*

Information Provisioning Enhancements

Information provisioning makes information available when and where it is needed. Oracle Database 10*g* Release 2 makes it is easier to bulk provision a large amount of information and to incrementally provision information using Streams.

See Also:

- Chapter 8, "Information Provisioning"
- Chapter 16, "Using Information Provisioning"

Oracle Database 10*g* Release 1 (10.1) New Features in Streams

The following sections describe the new features in Oracle Streams for Oracle Database 10*g* Release 1 (10.1):

- Streams Performance Improvements
- Streams Configuration and Manageability Enhancements
- Streams Replication Enhancements
- Streams Messaging Enhancements
- Rules Interface Enhancements

Streams Performance Improvements

Oracle Database 10*g* Release 1 includes performance improvements for most Streams operations. Specifically, the following Streams components have been improved to perform more efficiently and handle greater workloads:

- Capture processes
- Propagations
- Apply processes

This release also includes performance improvements for `ANYDATA` queue operations and **rule set** evaluations.

Streams Configuration and Manageability Enhancements

The following are Streams configuration manageability enhancements for Oracle Database 10*g* Release 1:

- Negative Rule Sets
- Downstream Capture
- Subset Rules for Capture and Propagation
- Streams Pool
- Access to Buffered Queue Information
- SYSAUX Tablespace Usage
- Ability to Add User-Defined Conditions to System-Created Rules

- Simpler Rule-Based Transformation Configuration and Administration
- Enqueue Destinations Upon Apply
- Execution Directives Upon Apply
- Support for Additional Datatypes
- Support for Index-Organized Tables
- Precommit Handlers
- Better Interoperation with Oracle Real Application Clusters
- Support for Function-Based Indexes and Descending Indexes
- Simpler Removal of Rule Sets When a Streams Client Is Dropped
- Simpler Removal of ANYDATA Queues
- Control Over Data Dictionary Builds in the Redo Log
- Additional Streams Data Dictionary Views and View Columns
- Copying and Moving Tablespaces
- Simpler Streams Administrator Configuration
- Streams Configuration Removal

Negative Rule Sets

Streams clients, which include **capture process**es, **propagation**s, **apply process**es, and **messaging client**s, can use two **rule set**s: a positive rule set and a negative rule set. Negative rule sets make it easier to discard specific changes so that they are not processed by a Streams client.

> **See Also:** Chapter 6, "How Rules Are Used in Streams"

Downstream Capture

A **capture process** can run on a database other than the **source database**. The redo log files from the source database are copied to the other database, called a **downstream database**, and the capture process captures changes in these redo log files at the downstream database.

> **See Also:**
>
> - "Downstream Capture" on page 2-13
> - "Creating a Capture Process" on page 11-2

Subset Rules for Capture and Propagation

You can use **subset rule**s for **capture process**es, **propagation**s, and **messaging client**s, as well as for **apply process**es.

> **See Also:** "Subset Rules" on page 6-17

Streams Pool

When Streams is used in a single database, memory is allocated from a pool in the System Global Area (SGA) called the Streams pool. The Streams pool contains **buffered queue**s and is used for internal communications during parallel capture and apply. Also, a new dynamic performance view, `V$STREAMS_POOL_ADVICE`, provides information that you can use to determine the best size for Streams pool.

See Also:

- "Streams Pool" on page 3-19
- "Setting Initialization Parameters Relevant to Streams" on page 10-4

Access to Buffered Queue Information

The following new dynamic performance views enable you to monitor **buffered queue**s:

- `V$BUFFERED_QUEUES`
- `V$BUFFERED_SUBSCRIBERS`
- `V$BUFFERED_PUBLISHERS`

See Also:

- "Buffered Queues" on page 3-21
- "Monitoring Buffered Queues" on page 21-5

SYSAUX Tablespace Usage

The default tablespace for LogMiner has been changed from the `SYSTEM` tablespace to the `SYSAUX` tablespace. When configuring a new database to run a **capture process**, you no longer need to relocate the LogMiner tables to a non-`SYSTEM` tablespace.

Ability to Add User-Defined Conditions to System-Created Rules

Some of the procedures that create **rule**s in the `DBMS_STREAMS_ADM` package include an `and_condition` parameter. This parameter enables you to add custom conditions to **system-created rule**s.

See Also: "System-Created Rules with Added User-Defined Conditions" on page 6-32

Simpler Rule-Based Transformation Configuration and Administration

A new procedure, `SET_RULE_TRANSFORM_FUNCTION` in the `DBMS_STREAMS_ADM` package, makes it easy to specify and administer **rule-based transformation**s.

See Also:

- Chapter 7, "Rule-Based Transformations"
- Chapter 15, "Managing Rule-Based Transformations"

Enqueue Destinations Upon Apply

A new procedure, `SET_ENQUEUE_DESTINATION` in the `DBMS_APPLY_ADM` package, makes it easy to specify a **destination queue** for **message**s that satisfy a particular **rule**. When a message satisfies such a rule in an **apply process rule set**, the apply process enqueues the message into the specified **queue**.

> **See Also:** "Specifying Message Enqueues by Apply Processes" on page 13-15

Execution Directives Upon Apply

A new procedure, `SET_EXECUTE` in the `DBMS_APPLY_ADM` package, enables you to specify that **apply process**es do not execute **message**s that satisfy a specific **rule**.

> **See Also:** "Specifying Execute Directives for Apply Processes" on page 13-16

Support for Additional Datatypes

Streams **capture process**es and **apply process**es now support the following additional datatypes:

- `NCLOB`
- `BINARY_FLOAT`
- `BINARY_DOUBLE`
- `LONG`
- `LONG RAW`

Logical change records (LCRs) containing these datatypes can also be propagated using **propagation**s.

> **See Also:**
>
> - "Datatypes Captured" on page 2-6
> - "Datatypes Applied" on page 4-8

Support for Index-Organized Tables

Streams **capture process**es and **apply process**es now support processing changes to index-organized tables.

> **See Also:**
>
> - "Types of DML Changes Captured" on page 2-8
> - *Oracle Streams Replication Administrator's Guide*

Precommit Handlers

You can use a new type of **apply handler** called a precommit handler to record information about commits processed by an apply process.

See Also:

- "Audit Commit Information for Messages Using Precommit Handlers" on page 4-6
- "Managing the Precommit Handler for an Apply Process" on page 13-13

Better Interoperation with Oracle Real Application Clusters

The following are specific enhancements that improve Streams interoperation with Oracle Real Application Clusters (RAC):

- Streams **capture process**es running in a RAC environment can capture changes in the online redo log as well as the archived redo log.
- If the owner instance for a **queue table** containing a **queue** used by a capture process or **apply process** becomes unavailable, then queue ownership is transferred automatically to another instance in the cluster and the capture process or apply process is restarted automatically (if it had been running).

See Also:

- "Streams Capture Processes and Oracle Real Application Clusters" on page 2-21
- "Streams Apply Processes and Oracle Real Application Clusters" on page 4-9

Support for Function-Based Indexes and Descending Indexes

Streams **capture process**es and **apply process**es now support processing changes to tables that use function-based indexes and descending indexes.

Simpler Removal of Rule Sets When a Streams Client Is Dropped

A new parameter, `drop_unused_rule_sets`, is added to the following procedures:

- `DROP_CAPTURE` in the `DBMS_CAPTURE_ADM` package
- `DROP_PROPAGATION` in the `DBMS_PROPAGATION_ADM` package
- `DROP_APPLY` in the `DBMS_APPLY_ADM` package

If you drop a **Streams client** using one of these procedures and set this parameter to `true`, then the procedure drops any **rule set**s, positive and negative, used by the specified Streams client if these rule sets are not used by any other Streams client. Streams clients include **capture process**es, **propagation**s, **apply process**es, and **messaging client**s. If this procedure drops a rule set, then this procedure also drops any **rule**s in the rule set that are not in another rule set.

See Also:

- "Dropping a Capture Process" on page 11-34
- "Dropping a Propagation" on page 12-14
- "Dropping an Apply Process" on page 13-26
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the procedures for dropping Streams clients

Simpler Removal of ANYDATA Queues

A new procedure, `REMOVE_QUEUE` in the `DBMS_STREAMS_ADM` package, enables you to remove an `ANYDATA` queue. This procedure also has a `cascade` parameter. When `cascade` is set to `true`, any Stream client that uses the **queue** is removed also.

See Also:

- "Removing an ANYDATA Queue" on page 12-6
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the REMOVE_QUEUE procedure

Control Over Data Dictionary Builds in the Redo Log

You can use the `BUILD` procedure in the `DBMS_CAPTURE_ADM` package to extract the data dictionary of the current database to the redo log. A **capture process** can use the extracted information in the redo log to create the **LogMiner data dictionary** for the capture process. This procedure also identifies a valid first system change number (SCN) value that can be used by the capture process. The **first SCN** for a capture process is the lowest SCN in the redo log from which a capture process can capture changes. In addition, you can reset the first SCN for a capture process to purge unneeded information in a LogMiner data dictionary.

See Also:

- "Capture Process Creation" on page 2-27
- "First SCN and Start SCN" on page 2-19
- "First SCN and Start SCN Specifications During Capture Process Creation" on page 2-33

Additional Streams Data Dictionary Views and View Columns

This release includes new Streams data dictionary views and new columns in Streams data dictionary views that existed in past releases.

See Also:

- Chapter 19, "Monitoring a Streams Environment" for an overview of the Streams data dictionary views and example queries
- *Oracle Streams Replication Administrator's Guide* for example queries that are useful in a Streams replication environment

Copying and Moving Tablespaces

The DBMS_STREAMS_TABLESPACE_ADM package provides administrative procedures for copying tablespaces between databases and moving tablespaces from one database to another. This package uses transportable tablespaces, Data Pump, and the DBMS_FILE_TRANSFER package.

> **See Also:** *Oracle Database PL/SQL Packages and Types Reference*

Simpler Streams Administrator Configuration

In this release, granting the DBA role to a Streams administrator is sufficient for most actions performed by the Streams administrator. In addition, a new package, DBMS_STREAMS_AUTH, provides procedures that make it easy for you to configure and manage a Streams administrator.

> **See Also:** "Configuring a Streams Administrator" on page 10-1

Streams Configuration Removal

A new procedure, REMOVE_STREAMS_CONFIGURATION in the DBMS_STREAMS_ADM package, enables you to remove the entire Streams configuration at a database.

> **See Also:** *Oracle Database PL/SQL Packages and Types Reference* for more information about the REMOVE_STREAMS_CONFIGURATION procedure

Streams Replication Enhancements

The following are Streams **replication** enhancements for Oracle Database 10*g* Release 1:

- Additional Supplemental Logging Options
- Additional Ways to Perform Instantiations
- New Data Dictionary Views for Schema and Global Instantiations
- Recursively Setting Schema and Global Instantiation SCN
- Access to Streams Client Information During LCR Processing
- Maintaining Tablespaces
- Control Over Comparing Old Values in Conflict Detection
- Extra Attributes in LCRs
- New Member Procedures and Functions for LCR Types
- A Generated Script to Migrate from Advanced Replication to Streams

Additional Supplemental Logging Options

For **database supplemental logging**, you can specify that all `FOREIGN KEY` columns in a database are supplementally logged, or that `ALL` columns in a database are supplementally logged. These new options are added to the `PRIMARY KEY` and `UNIQUE` options, which were available in past releases.

For table **supplemental logging**, you can specify the following options for log groups:

- `PRIMARY KEY`
- `FOREIGN KEY`
- `UNIQUE`
- `ALL`

These new options make it easier to specify and manage supplemental logging at a **source database** because you can specify supplemental logging without listing each column in a log group. If a table changes in the future, then the correct columns are logged automatically. For example, if you specify `FOREIGN KEY` for a table's log group, then the foreign key for a row is logged when the row is changed, even if the columns in the foreign key change in the future.

> **See Also:** *Oracle Streams Replication Administrator's Guide* for more information about supplemental logging in a Streams replication environment

Additional Ways to Perform Instantiations

In addition to original export/import, you can use Data Pump export/import, transportable tablespaces, and RMAN to perform Streams **instantiation**s.

> **See Also:** *Oracle Streams Replication Administrator's Guide* for more information about performing instantiations

New Data Dictionary Views for Schema and Global Instantiations

The following new data dictionary views enable you to determine which database objects have a set **instantiation SCN** at the schema and global level:

- `DBA_APPLY_INSTANTIATED_SCHEMAS`
- `DBA_APPLY_INSTANTIATED_GLOBAL`

Recursively Setting Schema and Global Instantiation SCN

A new `recursive` parameter in the `SET_SCHEMA_INSTANTIATION_SCN` and `SET_GLOBAL_INSTANTIATION_SCN` procedures enables you to set the **instantiation SCN** for a schema or database, respectively, and for all of the database objects in the schema or database.

> **See Also:**
>
> - *Oracle Streams Replication Administrator's Guide* for more information about performing **instantiation**s
> - *Oracle Database PL/SQL Packages and Types Reference* for more information about the `SET_SCHEMA_INSTANTIATION_SCN` and `SET_GLOBAL_INSTANTIATION_SCN` procedures

Access to Streams Client Information During LCR Processing

The DBMS_STREAMS package includes two new functions: GET_STREAMS_NAME and GET_STREAMS_TYPE. These functions return the name and type, respectively, of a **Streams client** that is processing an LCR. You can use these functions in **rule condition**s, **rule-based transformation**s, **apply handler**s, **error handler**s, and in a rule condition.

For example, if you use one error handler for multiple apply processes, then you can use the GET_STREAMS_NAME function to determine the name of the apply process that raised the error. Also, you can use the GET_STREAMS_TYPE function to instruct a **DML handler** to operate differently if it is processing **message**s from the error queue (ERROR_EXECUTION type) instead of the apply process **queue** (APPLY type).

See Also:

- "Managing an Error Handler" on page 13-18 for an example of an error handler that uses the GET_STREAMS_NAME function
- *Oracle Database PL/SQL Packages and Types Reference* for more information about these functions

Maintaining Tablespaces

You can use the MAINTAIN_SIMPLE_TABLESPACE procedure to configure Streams **replication** for a simple tablespace, and you can use the MAINTAIN_TABLESPACES procedure to configure Streams replication for a set of self-contained tablespaces. Both of these procedures are in the DBMS_STREAMS_ADM package. These procedures use transportable tablespaces, Data Pump, the DBMS_STREAMS_TABLESPACE_ADM package, and the DBMS_FILE_TRANSFER package to configure the environment.

See Also:

- *Oracle Streams Replication Administrator's Guide*
- *Oracle Database PL/SQL Packages and Types Reference*

Control Over Comparing Old Values in Conflict Detection

The COMPARE_OLD_VALUES procedure in the DBMS_APPLY_ADM package enables you to specify whether to compare old values of one or more columns in a row LCR with the current value of the corresponding columns at the **destination database** during apply.

See Also: *Oracle Database PL/SQL Packages and Types Reference*

Extra Attributes in LCRs

You can optionally use the INCLUDE_EXTRA_ATTRIBUTE procedure in the DBMS_CAPTURE_ADM package to instruct a **capture process** to include the following extra attributes in LCRs:

- row_id
- serial#
- session#
- thread#
- tx_name
- username

See Also: "Extra Information in LCRs" on page 2-5

New Procedure for Point-In-Time Recovery in a Streams Environment

The `GET_SCN_MAPPING` procedure in the `DBMS_STREAMS_ADM` package gets information about the SCN values to use for Streams capture and **apply process**es to recover transactions after point-in-time recovery is performed on a **source database** in a multiple-source Streams environment.

See Also: *Oracle Streams Replication Administrator's Guide*

New Member Procedures and Functions for LCR Types

You can use the following new member procedures and functions for LCR types:

- The `GET_COMMIT_SCN` member function returns the commit SCN of the transaction to which the current LCR belongs.
- The `GET_EXTRA_ATTRIBUTE` member function returns the value for the specified extra attribute in an LCR, and the `SET_EXTRA_ATTRIBUTE` member procedure enables you to set the value for the specified extra attribute in an LCR.
- The `GET_COMPATIBLE` member function returns the minimal database compatibility required to support an LCR.
- The `CONVERT_LONG_TO_LOB_CHUNK` member procedure converts `LONG` data in a row LCR into a `CLOB`, or converts `LONG RAW` data in a row LCR into a `BLOB`.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for more information about LCR types and the new member procedures and functions
- *Oracle Streams Replication Administrator's Guide* for an example of a **DML handler** that uses the `GET_COMMIT_SCN` member function
- "Rule Conditions that Instruct Streams Clients to Discard Unsupported LCRs" on page 6-42 for an example of a rule condition that uses the `GET_COMPATIBLE` member function

A Generated Script to Migrate from Advanced Replication to Streams

You can use the procedure `DBMS_REPCAT.STREAMS_MIGRATION` to generate a SQL*Plus script that migrates an existing Advanced Replication environment to a Streams environment.

See Also: *Oracle Streams Replication Administrator's Guide* for information about migrating from Advanced Replication to Streams

Streams Messaging Enhancements

The following are Streams messaging enhancements for Oracle Database 10*g* Release 1:

- Streams Messaging Client
- Simpler Enqueue and Dequeue of Messages
- Simpler Configuration of Rule-Based Dequeue or Apply of Messages
- Simpler Configuration of Rule-Based Propagations of Messages
- Simpler Configuration of Message Notifications

> **See Also:** *Oracle Streams Advanced Queuing User's Guide and Reference* for more information about Streams messaging enhancements

Streams Messaging Client

A messaging client is a new type of **Streams client** that enables users and applications to dequeue **message**s from an `ANYDATA` queue based on **rule**s. You can create a messaging client by specifying `dequeue` for the `streams_type` parameter in certain procedures in the `DBMS_STREAMS_ADM` package.

> **See Also:**
>
> - Chapter 3, "Streams Staging and Propagation"
> - "Message Rule Example" on page 6-27
> - "Configuring a Messaging Client and Message Notification" on page 12-18
> - *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_STREAMS_ADM` package

Simpler Enqueue and Dequeue of Messages

A new package, `DBMS_STREAMS_MESSAGING`, provides an easy interface for enqueuing **message**s into and dequeuing messages from an `ANYDATA` queue.

> **See Also:**
>
> - "Configuring a Messaging Client and Message Notification" on page 12-18
> - *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_STREAMS_MESSAGING` package

Simpler Configuration of Rule-Based Dequeue or Apply of Messages

A new procedure, `ADD_MESSAGE_RULE` in the `DBMS_STREAMS_ADM` package, enables you to configure **messaging client**s and **apply process**es, and it enables you to create the **rule**s for **user-enqueued message**s that control the behavior of these messaging clients and apply processes.

> **See Also:**
>
> - "Message Rules" on page 6-26
> - *Oracle Database PL/SQL Packages and Types Reference* for more information about the `ADD_MESSAGE_RULE` procedure

Simpler Configuration of Rule-Based Propagations of Messages

A new procedure, `ADD_MESSAGE_PROPAGATION_RULE` in the `DBMS_STREAMS_ADM` package, enables you to configure **propagation**s and create **rule**s for propagations that propagate **user-enqueued message**s.

> **See Also:** *Oracle Database PL/SQL Packages and Types Reference* for more information about the `ADD_MESSAGE_PROPAGATION_RULE` procedure

Simpler Configuration of Message Notifications

A new procedure, `SET_MESSAGE_NOTIFICATION` in the `DBMS_STREAMS_ADM` package, enables you to configure **message** notifications that are sent when a Streams **messaging client** dequeues messages. The notification can be sent to an email address, a URL, or a PL/SQL procedure.

> **See Also:**
>
> - "Configuring a Messaging Client and Message Notification" on page 12-18
> - *Oracle Database PL/SQL Packages and Types Reference* for more information about the `SET_MESSAGE_NOTIFICATION` procedure

Rules Interface Enhancements

The following are **rule**s interface enhancements for Oracle Database 10*g* Release 1:

- Iterative Evaluation Results
- New Dynamic Performance Views for Rule Sets and Rule Evaluations

Iterative Evaluation Results

During **rule set** evaluation, a client now can specify that evaluation results are sent iteratively, instead of in a complete list at one time. The `EVALUATE` procedure in the `DBMS_RULE` package includes the following two new parameters that enable you specify that evaluation results are sent iteratively: `true_rules_interator` and `maybe_rules_iterator`.

In addition, a new procedure in the `DBMS_RULE` package, `GET_NEXT_HIT`, returns the next **rule** that evaluated to `TRUE` from a true rules iterator, or returns the next rule that evaluated to `MAYBE` from a maybe rules iterator. Also, the new `CLOSE_ITERATOR` procedure in the `DBMS_RULE` package enables you to close an open iterator.

> **See Also:**
>
> - "Rule Set Evaluation" on page 5-10
> - Chapter 28, "Rule-Based Application Example" for examples that use iterative evaluation results
> - *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_RULE` package

New Dynamic Performance Views for Rule Sets and Rule Evaluations

You can use the following new dynamic performance views to monitor **rule set**s and **rule** evaluations:

- `V$RULE_SET_AGGREGATE_STATS`
- `V$RULE_SET`
- `V$RULE`

See Also: Chapter 23, "Monitoring Rules"

Part I

Streams Concepts

This part describes conceptual information about Streams and contains the following chapters:

1

Introduction to Streams

This chapter briefly describes the basic concepts and terminology related to Oracle Streams. These concepts are described in more detail in other chapters in this book and in the *Oracle Streams Replication Administrator's Guide*.

The following topics introduce Oracle Streams and the Streams page in Oracle Enterprise Manager:

This chapter contains these topics:

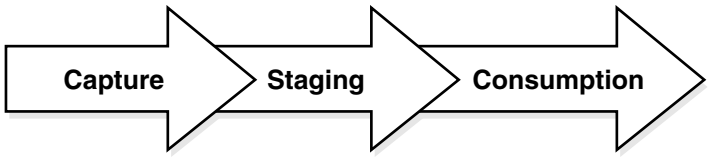
Overview of Streams

Oracle Streams enables information sharing. Using Oracle Streams, each unit of shared information is called a **message**, and you can share these messages in a stream. The stream can propagate information within a database or from one database to another. The stream routes specified information to specified destinations. The result is a feature that provides greater functionality and flexibility than traditional solutions for capturing and managing messages, and sharing the messages with other databases and applications. Streams provides the capabilities needed to build and operate distributed enterprises and applications, data warehouses, and high availability solutions. You can use all of the capabilities of Oracle Streams at the same time. If your needs change, then you can implement a new capability of Streams without sacrificing existing capabilities.

Using Oracle Streams, you control what information is put into a stream, how the stream flows or is routed from database to database, what happens to messages in the stream as they flow into each database, and how the stream terminates. By configuring specific capabilities of Streams, you can address specific requirements. Based on your specifications, Streams can capture, stage, and manage messages in the database automatically, including, but not limited to, data manipulation language (DML) changes and data definition language (DDL) changes. You can also put user-defined messages into a stream, and Streams can propagate the information to other databases or applications automatically. When messages reach a destination, Streams can consume them based on your specifications.

Figure 1–1 shows the Streams information flow.

Figure 1–1 Streams Information Flow



What Can Streams Do?

The following sections provide an overview of what Streams can do.

- Capture Messages at a Database
- Stage Messages in a Queue
- Propagate Messages from One Queue to Another
- Consume Messages
- Other Capabilities of Streams

Capture Messages at a Database

A **capture process** can capture database events, such as changes made to tables, schemas, or an entire database. Such changes are recorded in the redo log for a database, and a capture process captures changes from the redo log and formats each captured change into a **message** called a **logical change record (LCR)**. The **rule**s used by a capture process determine which changes it captures, and these captured changes are called **captured message**s.

The database where changes are generated in the redo log is called the **source database**. A capture process can capture changes locally at the source database, or it can capture changes remotely at a **downstream database**. A capture process enqueues logical change records (LCRs) into a **queue** that is associated with it. When a capture process captures messages, it is sometimes referred to as **implicit capture**.

Users and applications can also enqueue messages into a queue manually. These messages are called **user-enqueued message**s, and they can be LCRs or messages of a user-defined type called **user message**s. When users and applications enqueue messages into a queue manually, it is sometimes referred to as **explicit capture**.

Stage Messages in a Queue

Messages are stored (or staged) in a **queue**. These **message**s can be **captured message**s or **user-enqueued message**s. A capture process enqueues messages into a **ANYDATA queue**. An `ANYDATA` queue can stage messages of different types. Users and applications can enqueue messages into an `ANYDATA` queue or into a **typed queue**. A typed queue can stage messages of one specific type only.

Propagate Messages from One Queue to Another

Streams **propagation**s can propagate **message**s from one **queue** to another. These queues can be in the same database or in different databases. Rules determine which messages are propagated by a propagation.

Consume Messages

A **message** is consumed when it is dequeued from a **queue**. An **apply process** can dequeue messages from a queue implicitly. A user, application, or **messaging client** can dequeue messages explicitly. The database where messages are consumed is called the **destination database**. In some configurations, the **source database** and the destination database can be the same.

Rules determine which messages are dequeued and processed by an apply process. An apply process can apply messages directly to database objects or pass messages to custom PL/SQL subprograms for processing.

Rules determine which messages are dequeued by a **messaging client**. A messaging client dequeues messages when it is invoked by an application or a user.

Other Capabilities of Streams

Other capabilities of Streams include the following:

- **directed network**s
- automatic **conflict** detection and **conflict resolution**
- **rule-based transformation**s
- **heterogeneous information sharing**

These capabilities are discussed briefly later in this chapter and in detail later in this document and in the *Oracle Streams Replication Administrator's Guide*.

What Are the Uses of Streams?

The following sections briefly describe some of the reasons for using Streams. In some cases, Streams components provide infrastructure for various features of Oracle.

- Message Queuing
- Data Replication
- Event Management and Notification
- Data Warehouse Loading
- Data Protection
- Database Availability During Upgrade and Maintenance Operations

Message Queuing

Oracle Streams Advanced Queuing (AQ) enables user applications to enqueue **message**s into a **queue**, propagate messages to subscribing queues, notify user applications that messages are ready for **consumption**, and dequeue messages at the destination. A queue can be configured to stage messages of a particular type only, or a queue can be configured as an `ANYDATA` queue. Messages of almost any type can be wrapped in an `ANYDATA` wrapper and staged in `ANYDATA` queues. AQ supports all the standard features of message queuing systems, including multiconsumer queues, publish and subscribe, content-based routing, Internet propagation, transformations, and gateways to other messaging subsystems.

You can create a queue at a database, and applications can enqueue messages into the queue explicitly. Subscribing applications or **messaging client**s can dequeue messages directly from this queue. If an application is remote, then a queue can be created in a remote database that subscribes to messages published in the source queue. The destination application can dequeue messages from the remote queue. Alternatively, the destination application can dequeue messages directly from the source queue using a variety of standard protocols.

> **See Also:** *Oracle Streams Advanced Queuing User's Guide and Reference* for more information about AQ

Data Replication

Streams can capture DML and DDL changes made to database objects and replicate those changes to one or more other databases. A Streams **capture process** captures changes made to **source database** objects and formats them into LCRs, which can be propagated to **destination database**s and then applied by Streams **apply process**es.

The destination databases can allow DML and DDL changes to the same database objects, and these changes might or might not be propagated to the other databases in the environment. In other words, you can configure a Streams environment with one database that propagates changes, or you can configure an environment where changes are propagated between databases bidirectionally. Also, the tables for which data is shared do not need to be identical copies at all databases. Both the structure and the contents of these tables can differ at different databases, and the information in these tables can be shared between these databases.

> **See Also:** *Oracle Streams Replication Administrator's Guide* for more information using Streams for **replication**

Event Management and Notification

Business events are valuable communications between applications or organizations. An application can enqueue **message**s that represent events into a **queue** explicitly, or a Streams **capture process** can capture database events and encapsulate them into messages called LCRs. These **captured message**s can be the results of DML or DDL changes. Propagations can propagate messages in a stream through multiple queues. Finally, a user application can dequeue messages explicitly, or a Streams **apply process** can dequeue messages implicitly. An apply process can reenqueue these messages explicitly into the same queue or a different queue if necessary.

You can configure queues to retain explicitly-enqueued messages after **consumption** for a specified period of time. This capability enables you to use Advanced Queuing (AQ) as a business event management system. AQ stores all messages in the database in a transactional manner, where they can be automatically audited and tracked. You can use this audit trail to extract intelligence about the business operations.

Streams capture processes, **propagation**s, apply processes, and **messaging client**s perform actions based on **rule**s. You specify which events are captured, propagated, applied, and dequeued using rules, and a built-in **rules engine** evaluates events based on these rules. The ability to capture events and propagate them to relevant consumers based on rules means that you can use Streams for event notification. Messages representing events can be staged in a queue and dequeued explicitly by a messaging client or an application, and then actions can be taken based on these events, which can include an email notification, or passing the message to a wireless gateway for transmission to a cell phone or pager.

See Also:

- Chapter 3, "Streams Staging and Propagation", Chapter 12, "Managing Staging and Propagation", and *Oracle Streams Advanced Queuing User's Guide and Reference* for more information about explicitly enqueuing and dequeuing messages
- Chapter 27, "Single-Database Capture and Apply Example" for a sample environment that explicitly dequeues messages

Data Warehouse Loading

Data warehouse loading is a special case of data **replication**. Some of the most critical tasks in creating and maintaining a data warehouse include refreshing existing data, and adding new data from the operational databases. Streams components can capture changes made to a production system and send those changes to a staging database or directly to a data warehouse or operational data store. Streams capture of redo data avoids unnecessary overhead on the production systems. Support for data transformations and user-defined apply procedures enables the necessary flexibility to reformat data or update warehouse-specific data fields as data is loaded. In addition, Change Data Capture uses some of the components of Streams to identify data that has changed so that this data can be loaded into a data warehouse.

See Also: *Oracle Database Data Warehousing Guide* for more information about data warehouses

Data Protection

One solution for data protection is to create a local or remote copy of a production database. In the event of human error or a catastrophe, the copy can be used to resume processing. You can use Streams to configure flexible high availability environments.

In addition, you can use Oracle Data Guard, a data protection feature that uses some of the same infrastructure as Streams, to create and maintain a logical standby database, which is a logically equivalent standby copy of a production database. As in the case of Streams **replication**, a **capture process** captures changes in the redo log and formats these changes into LCRs. These LCRs are applied at the standby databases. The standby databases are fully open for read/write and can include specialized indexes or other database objects. Therefore, these standby databases can be queried as updates are applied.

It is important to move the updates to the remote site as soon as possible with a logical standby database. Doing so ensures that, in the event of a failure, lost transactions are minimal. By directly and synchronously writing the redo logs at the remote database, you can achieve no data loss in the event of a disaster. At the standby system, the changes are captured and directly applied to the standby database with an **apply process**.

See Also:

- Chapter 9, "Streams High Availability Environments"
- *Oracle Data Guard Concepts and Administration* for more information about logical standby databases

Database Availability During Upgrade and Maintenance Operations

You can use the features of Oracle Streams to achieve little or no database down time during database upgrade and maintenance operations. Maintenance operations include migrating a database to a different platform, migrating a database to a different character set, modifying database schema objects to support upgrades to user-created applications, and applying an Oracle software patch.

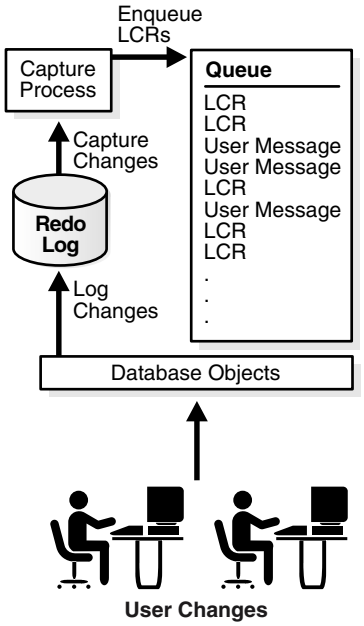
See Also:

- Appendix B, "Online Database Upgrade with Streams"
- Appendix C, "Online Database Maintenance with Streams"

Overview of the Capture Process

Changes made to database objects in an Oracle database are logged in the redo log to guarantee recoverability in the event of user error or media failure. A **capture process** is an Oracle background process that scans the database redo log to capture DML and DDL changes made to database objects. A capture process formats these changes into **message**s called LCRs and enqueues them into a **queue**. There are two types of LCRs: **row LCRs** contain information about a change to a row in table resulting from a DML operation, and **DDL LCRs** contain information about a DDL change to a database object. Rules determine which changes are captured. Figure 1–2 shows a capture process capturing LCRs.

Figure 1–2 Capture Process



You can configure change capture locally at a **source database** or remotely at a downstream database. A **local capture process** runs at the source database and captures changes from the local source database redo log. The following types of configurations are possible for a **downstream capture process**:

- A **real-time downstream capture** configuration means that the log writer process (LGWR) at the source database sends redo data from the online redo log to the downstream database. At the downstream database, the redo data is stored in the **standby redo log**, and the capture process captures changes from the standby redo log.
- An **archived-log downstream capture** configuration means that archived redo log files from the source database are copied to the downstream database, and the capture process captures changes in these archived redo log files.

Note: A capture process does not capture some types of DML and DDL changes, and it does not capture changes made in the `SYS`, `SYSTEM`, or `CTXSYS` schemas.

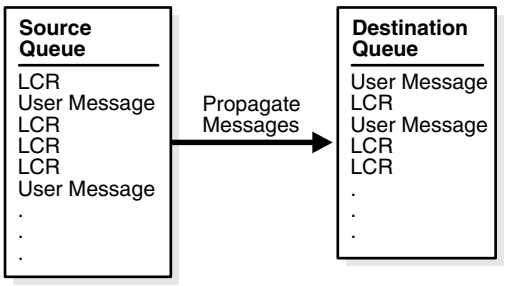
See Also: Chapter 2, "Streams Capture Process" for more information about capture processes and for detailed information about which DML and DDL statements are captured by a capture process

Overview of Message Staging and Propagation

Streams uses **queue**s to stage **message**s for propagation or **consumption**. Propagations send messages from one queue to another, and these queues can be in the same database or in different databases. The queue from which the messages are propagated is called the **source queue**, and the queue that receives the messages is called the **destination queue**. There can be a one-to-many, many-to-one, or many-to-many relationship between source and destination queues.

Messages that are staged in a queue can be consumed by an **apply process**, a **messaging client**, or an application. Rules determine which messages are propagated by a **propagation**. Figure 1–3 shows propagation from a source queue to a destination queue.

Figure 1–3 Propagation from a Source Queue to a Destination Queue



See Also: Chapter 3, "Streams Staging and Propagation" for more information about staging and propagation

Overview of Directed Networks

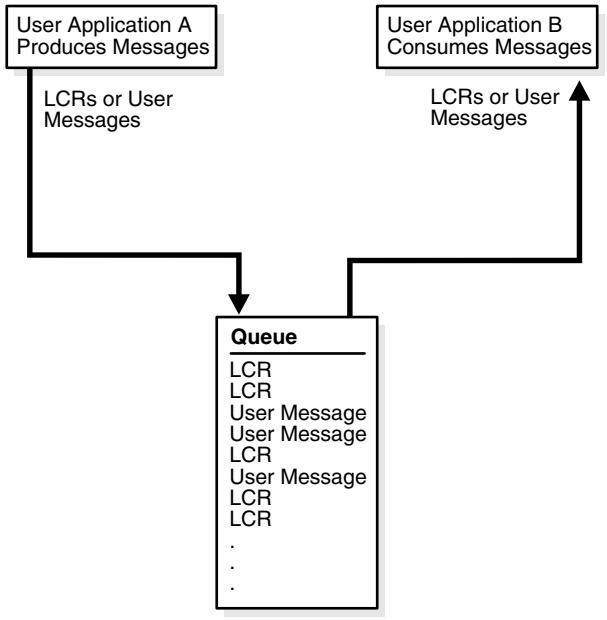
Streams enables you to configure an environment in which changes are shared through **directed networks**. In a directed network, propagated **message**s pass through one or more intermediate databases before arriving at a **destination database** where they are consumed. The messages might or might not be consumed at an intermediate database in addition to the destination database. Using Streams, you can choose which messages are propagated to each destination database, and you can specify the route messages will traverse on their way to a destination database.

See Also: "Directed Networks" on page 3-7

Explicit Enqueue and Dequeue of Messages

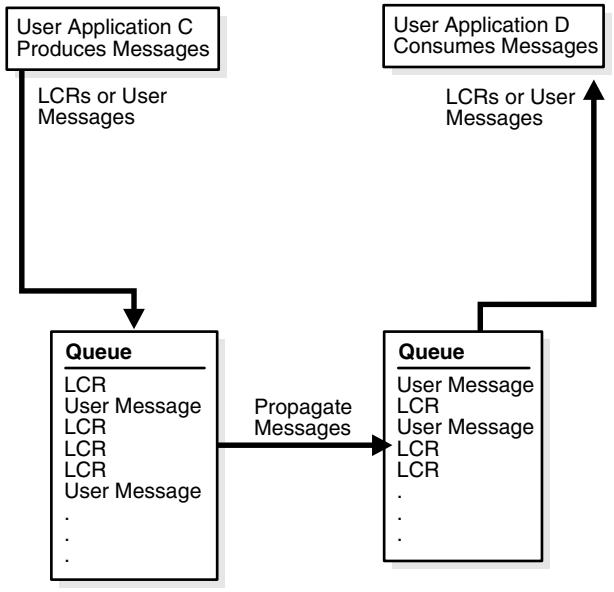
User applications can enqueue **message**s into a **queue** explicitly. The user applications can format these **user-enqueued message**s as LCRs or **user message**s, and an **apply process**, a **messaging client**, or a user application can consume these messages. Messages that were enqueued explicitly into a queue can be propagated to another queue or explicitly dequeued from the same queue. Figure 1–4 shows explicit enqueue of messages into and dequeue of messages from the same queue.

Figure 1–4 Explicit Enqueue and Dequeue of Messages in a Single Queue



When messages are propagated between queues, messages that were enqueued explicitly into a **source queue** can be dequeued explicitly from a **destination queue** by a messaging client or user application. These messages can also be processed by an apply process. Figure 1–5 shows explicit enqueue of messages into a **source queue**, **propagation** to a destination queue, and then explicit dequeue of messages from the destination queue.

Figure 1–5 Explicit Enqueue, Propagation, and Dequeue of Messages



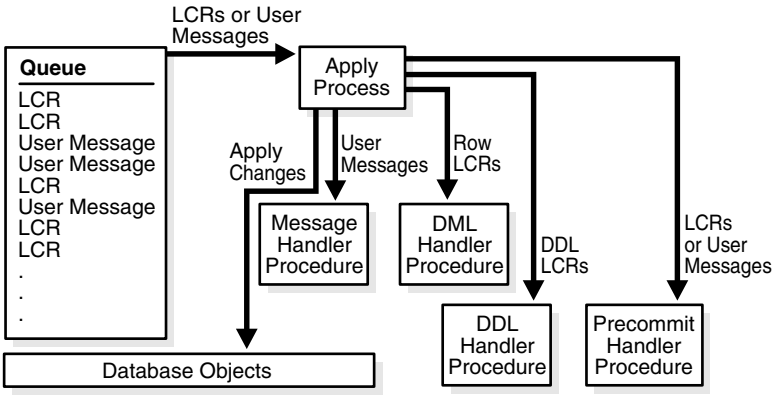
See Also: "ANYDATA Queues and User Messages" on page 3-11 for more information about explicit enqueue and dequeue of messages

Overview of the Apply Process

An apply process is an Oracle background process that dequeues **message**s from a **queue** and either applies each message directly to a database object or passes the message as a parameter to a user-defined procedure called an **apply handler**. Apply handlers include **message handler**s, **DML handler**s, **DDL handler**, **precommit handler**s, and **error handler**s.

Typically, an apply process applies messages to the local database where it is running, but, in a heterogeneous database environment, it can be configured to apply messages at a remote non-Oracle database. Rules determine which messages are dequeued by an apply process. Figure 1–6 shows an apply process processing LCRs and **user message**s.

Figure 1–6 Apply Process

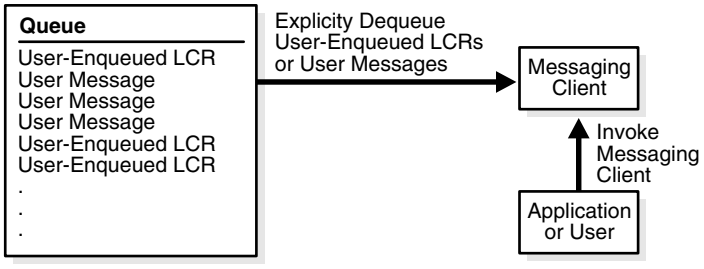


See Also: Chapter 4, "Streams Apply Process"

Overview of the Messaging Client

A messaging client consumes **user-enqueued message**s when it is invoked by an application or a user. Rules determine which user-enqueued messages are dequeued by a messaging client. These user-enqueued messages can be LCRs or **user message**s. Figure 1–7 shows a messaging client dequeuing user-enqueued messages.

Figure 1–7 Messaging Client



See Also: "Messaging Clients" on page 3-10

Overview of Automatic Conflict Detection and Resolution

An **apply process** detects conflicts automatically when directly applying LCRs in a **replication** environment. A **conflict** is a mismatch between the old values in an LCR and the expected data in a table. Typically, a conflict results when the same row in the **source database** and **destination database** is changed at approximately the same time.

When a conflict occurs, you need a mechanism to ensure that the conflict is resolved in accordance with your business rules. Streams offers a variety of prebuilt conflict handlers. Using these prebuilt handlers, you can define a **conflict resolution** system for each of your databases that resolves conflicts in accordance with your business rules. If you have a unique situation that prebuilt conflict resolution handlers cannot resolve, then you can build your own conflict resolution handlers.

If a conflict is not resolved, or if a handler procedure raises an error, then all **message**s in the transaction that raised the error are saved in the error queue for later analysis and possible reexecution.

See Also: *Oracle Streams Replication Administrator's Guide*

Overview of Rules

Streams enables you to control which information to share and where to share it using **rules**. A rule is specified as a condition that is similar to the condition in the `WHERE` clause of a SQL query.

A rule consists of the following components:

- The **rule condition** combines one or more **expression**s and conditions and returns a Boolean value, which is a value of `TRUE`, `FALSE`, or `NULL` (unknown), based on an event.
- The **evaluation context** defines external data that can be referenced in rule conditions. The external data can either exist as external variables, as table data, or both.
- The **action context** is optional information associated with a rule that is interpreted by the client of the **rules engine** when the rule is evaluated.

You can group related rules together into **rule sets**. In Streams, rule sets can be positive or negative.

For example, the following rule condition can be used for a rule in Streams to specify that the schema name that owns a table must be `hr` and that the table name must be `departments` for the condition to evaluate to `TRUE`:

```
:dml.get_object_owner() = 'HR' AND :dml.get_object_name() = 'DEPARTMENTS'
```

The `:dml` variable is used in rule conditions for row LCRs. In a Streams environment, a rule with this condition can be used in the following ways:

- If the rule is in a **positive rule set** for a **capture process**, then it instructs the capture process to capture row changes that result from DML changes to the `hr.departments` table. If the rule is in a **negative rule set** for a capture process, then it instructs the capture process to discard DML changes to the `hr.departments` table.

- If the rule is in a positive rule set for a **propagation**, then it instructs the propagation to propagate LCRs that contain row changes to the `hr.departments` table. If the rule is in a negative rule set for a propagation, then it instructs the propagation to discard LCRs that contain row changes to the `hr.departments` table.
- If the rule is in a positive rule set for an **apply process**, then it instructs the apply process to apply LCRs that contain row changes to the `hr.departments` table. If the rule is in a negative rule set for an apply process, then it instructs the apply process to discard LCRs that contain row changes to the `hr.departments` table.
- If the rule is in a positive rule set for a **messaging client**, then it instructs the messaging client to dequeue LCRs that contain row changes to the `hr.departments` table. If the rule is in a negative rule set for a messaging client, then it instructs the messaging client to discard LCRs that contain row changes to the `hr.departments` table.

Streams performs tasks based on rules. These tasks include capturing **message**s with a capture process, propagating messages with a propagation, applying messages with an apply process, dequeuing messages with a messaging client, and discarding messages.

See Also:

- Chapter 5, "Rules"
- Chapter 6, "How Rules Are Used in Streams"

Overview of Rule-Based Transformations

A **rule-based transformation** is any modification to a **message** that results when a **rule** in a **positive rule set** evaluates to `TRUE`. There are two types of rule-based transformations: declarative and custom.

Declarative rule-based transformations cover a set of common transformation scenarios for row LCRs, including renaming a schema, renaming a table, adding a column, renaming a column, and deleting a column. You specify (or declare) such a transformation using a procedure in the `DBMS_STREAMS_ADM` package. Streams performs declarative transformations internally, without invoking PL/SQL.

A **custom rule-based transformation** requires a user-defined PL/SQL function to perform the transformation. Streams invokes the PL/SQL function to perform the transformation. A custom rule-based transformation can modify either **captured message**s or **user-enqueued message**s, and these messages can be LCRs or **user message**s. For example, a custom rule-based transformation can change the datatype of a particular column in an LCR.

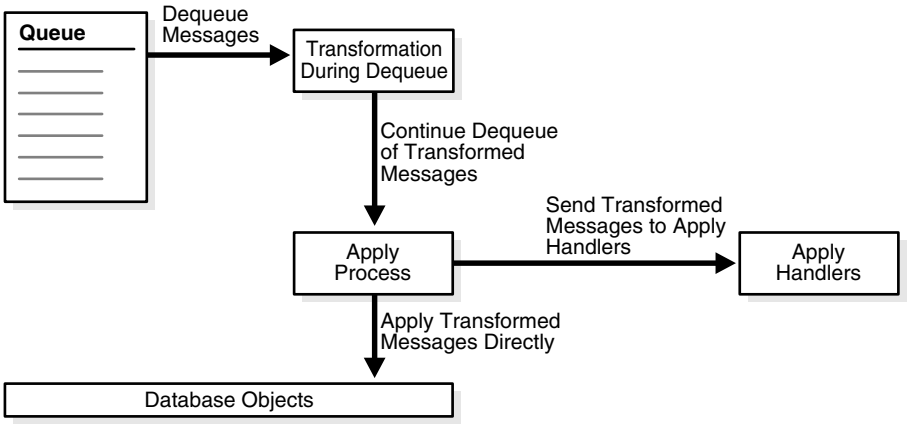
To specify a custom rule-based transformation, use the `DBMS_STREAMS_ADM.SET_RULE_TRANSFORM_FUNCTION` procedure. The transformation function takes as input an `ANYDATA` object containing a message and returns an `ANYDATA` object containing the transformed message. For example, a transformation can use a PL/SQL function that takes as input an `ANYDATA` object containing an LCR with a `NUMBER` datatype for a column and returns an `ANYDATA` object containing an LCR with a `VARCHAR2` datatype for the same column.

Either type of rule-based transformation can occur at the following times:

- During enqueue of a message by a **capture process**, which can be useful for formatting a message in a manner appropriate for all **destination database**s
- During propagation of a message, which can be useful for transforming a message before it is sent to a specific remote site
- During dequeue of a message by an **apply process** or **messaging client**, which can be useful for formatting a message in a manner appropriate for a specific destination database

When a transformation is performed during apply, an apply process can apply the transformed message directly or send the transformed message to an **apply handler** for processing. Figure 1–8 shows a rule-based transformation during apply.

Figure 1–8 Transformation During Apply



Note:

- A rule must be in a positive rule set for its rule-based transformation to be invoked. A rule-based transformation specified for a rule in a **negative rule set** is ignored by **capture process**es, **propagation**s, **apply process**es, and **messaging client**s.
- Throughout this document, "rule-based transformation" is used when the text applies to both declarative and custom rule-based transformations. This document distinguishes between the two types of rule-based transformations when necessary.

See Also: Chapter 7, "Rule-Based Transformations"

Overview of Streams Tags

Every redo entry in the redo log has a **tag** associated with it. The datatype of the tag is `RAW`. By default, when a user or application generates redo entries, the value of the tag is `NULL` for each redo entry, and a `NULL` tag consumes no space in the redo entry. The size limit for a tag value is 2000 bytes.

In Streams, **rule**s can have conditions relating to tag values to control the behavior of **Streams client**s. For example, a tag can be used to determine whether an LCR contains a change that originated in the local database or at a different database, so that you can avoid **change cycling** (sending an LCR back to the database where it originated). Also, a tag can be used to specify the set of **destination database**s for each LCR. Tags can be used for other LCR tracking purposes as well.

You can specify Streams tags for redo entries generated by a certain session or by an **apply process**. These tags then become part of the LCRs captured by a **capture process**. Typically, tags are used in Streams **replication** environments, but you can use them whenever it is necessary to track database changes and LCRs.

> **See Also:** *Oracle Streams Replication Administrator's Guide* for more information about Streams tags

Overview of Heterogeneous Information Sharing

In addition to information sharing between Oracle databases, Streams supports information sharing between Oracle databases and non-Oracle databases. The following sections contain an overview of this support.

> **See Also:** *Oracle Streams Replication Administrator's Guide* for more information about heterogeneous information sharing with Streams

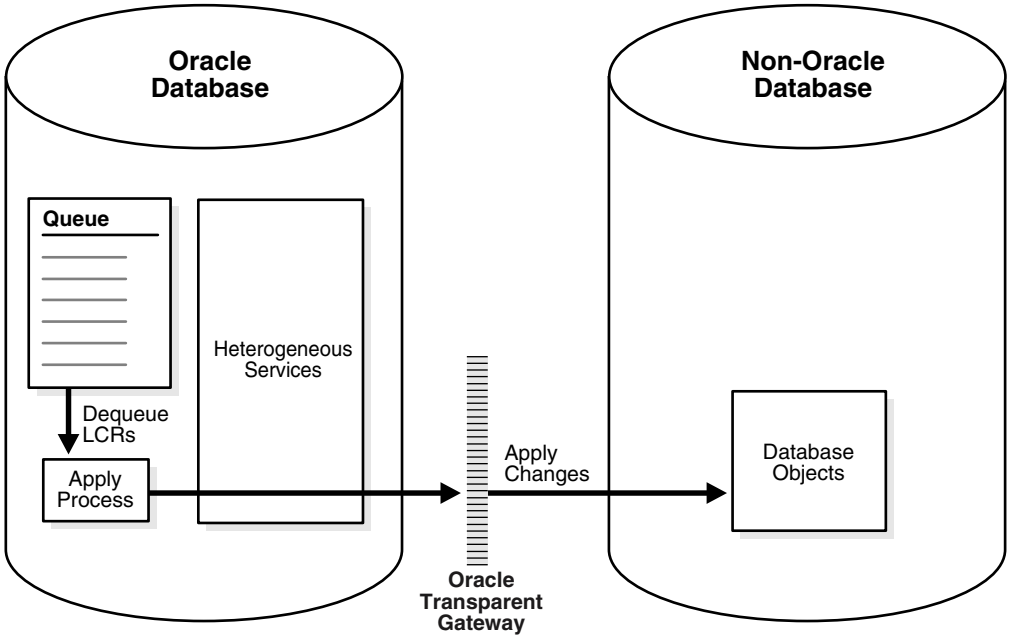
Overview of Oracle to Non-Oracle Data Sharing

If an Oracle database is the source and a non-Oracle database is the destination, then the non-Oracle database destination lacks the following Streams mechanisms:

- A **queue** to receive **message**s
- An **apply process** to dequeue and apply messages

To share DML changes from an Oracle **source database** with a non-Oracle **destination database**, the Oracle database functions as a proxy and carries out some of the steps that would normally be done at the destination database. That is, the messages intended for the non-Oracle destination database are dequeued in the Oracle database itself, and an apply process at the Oracle database uses Heterogeneous Services to apply the messages to the non-Oracle database across a network connection through a gateway. Figure 1–9 shows an Oracle databases sharing data with a non-Oracle database.

Figure 1–9 Oracle to Non-Oracle Heterogeneous Data Sharing

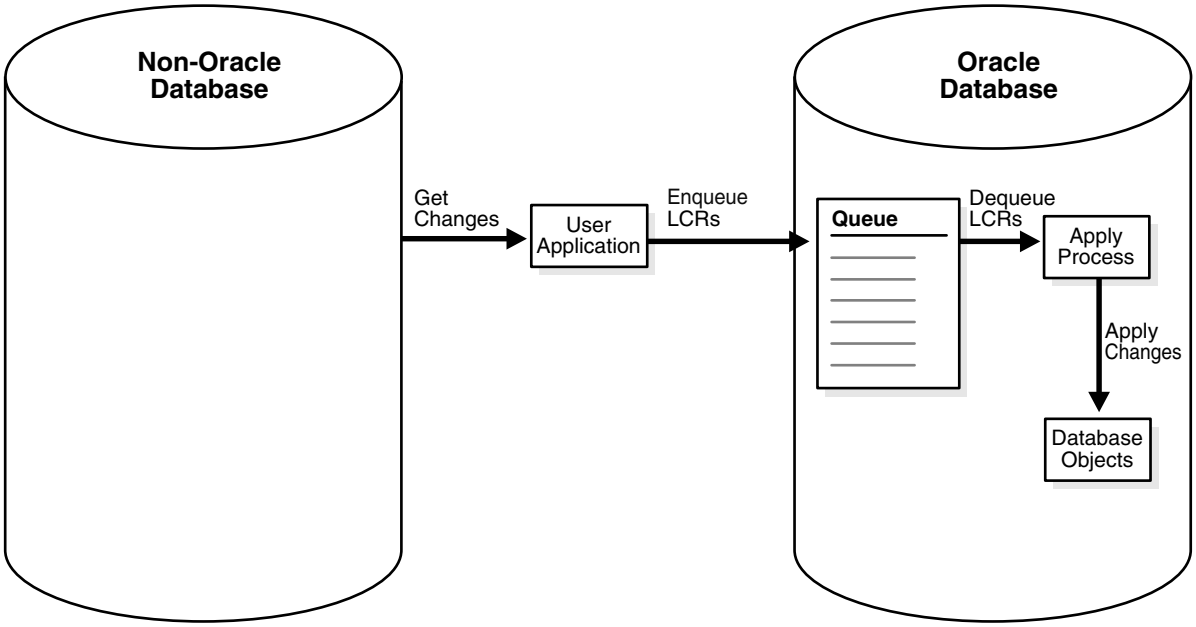


See Also: *Oracle Database Heterogeneous Connectivity Administrator's Guide* for more information about Heterogeneous Services

Overview of Non-Oracle to Oracle Data Sharing

To capture and propagate changes from a non-Oracle database to an Oracle database, a custom application is required. This application gets the changes made to the non-Oracle database by reading from transaction logs, using triggers, or some other method. The application must assemble and order the transactions and must convert each change into an LCR. Next, the application must enqueue the LCRs into a **queue** in an Oracle database by using the PL/SQL interface, where they can be processed by an **apply process**. Figure 1–10 shows a non-Oracle databases sharing data with an Oracle database.

Figure 1–10 Non-Oracle to Oracle Heterogeneous Data Sharing



Example Streams Configurations

Figure 1–11 shows how Streams might be configured to share information within a single database, while Figure 1–12 shows how Streams might be configured to share information between two different databases.

Figure 1–11 Streams Configuration in a Single Database

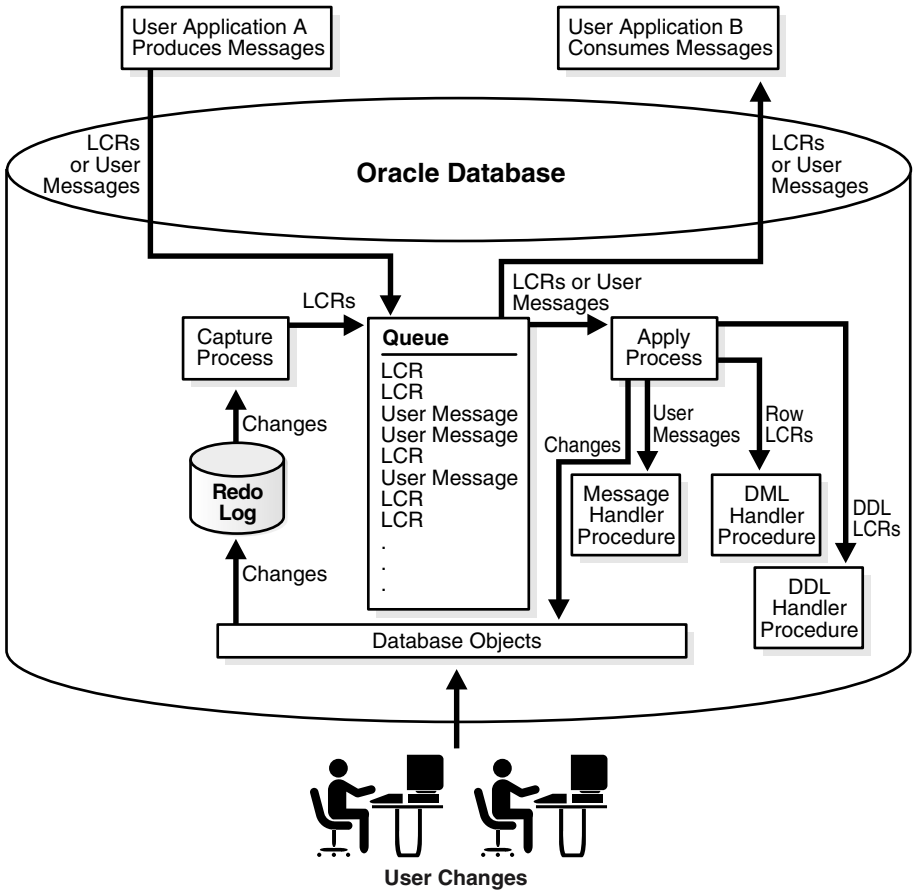
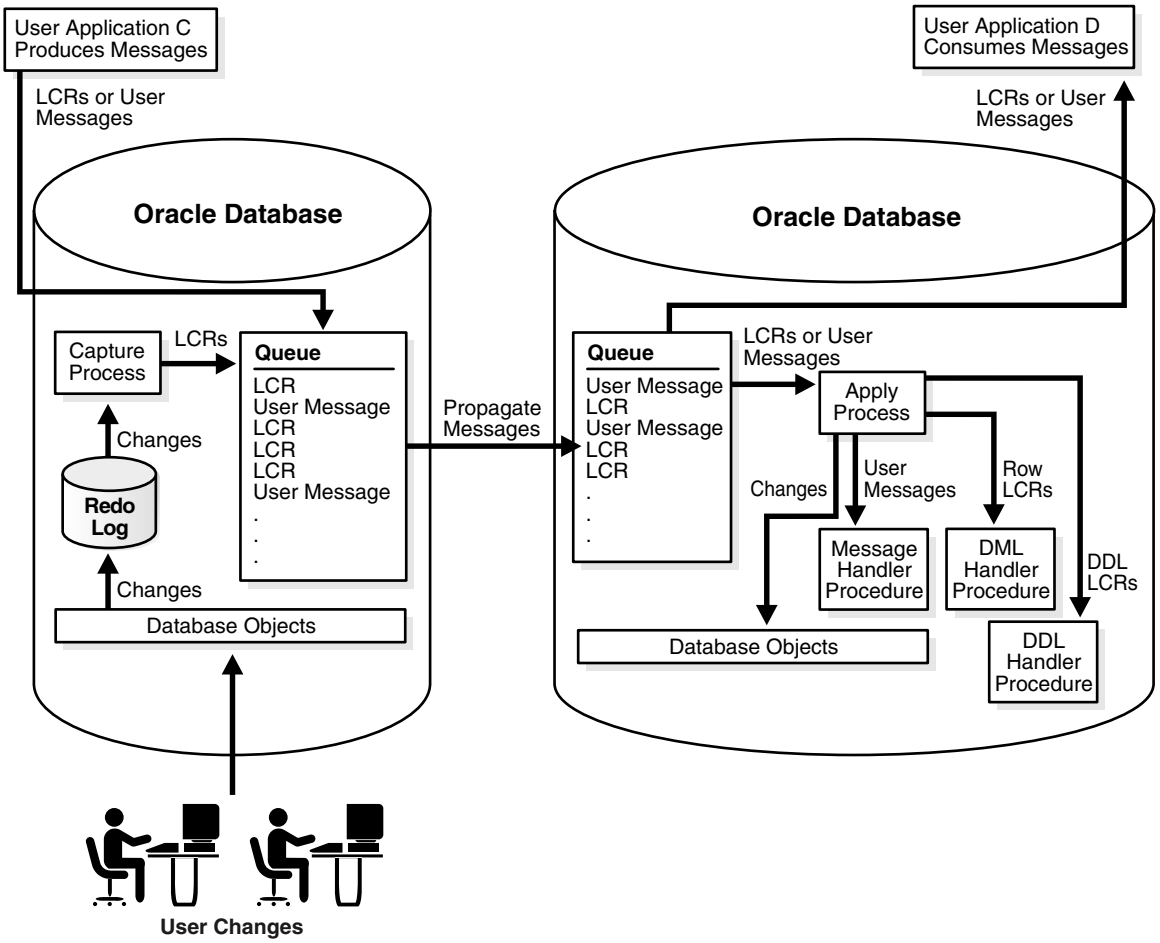


Figure 1–12 Streams Configuration Sharing Information Between Databases



Administration Tools for a Streams Environment

Several tools are available for configuring, administering, and monitoring your Streams environment. Oracle-supplied PL/SQL packages are the primary configuration and management tools, and the Streams tool in Oracle Enterprise Manager provides some configuration, administration, and monitoring capabilities to help you manage your environment. Additionally, Streams data dictionary views keep you informed about your Streams environment.

Oracle-Supplied PL/SQL Packages

The following Oracle-supplied PL/SQL packages contain procedures and functions for configuring and managing a Streams environment.

> **See Also:** *Oracle Database PL/SQL Packages and Types Reference* for more information about these packages

DBMS_STREAMS_ADM Package

The DBMS_STREAMS_ADM package provides an administrative interface for adding and removing simple **rule**s for **capture process**es, **propagation**s, and **apply process**es at the table, schema, and database level. This package also enables you to add rules that control which **message**s a propagation propagates and which messages a **messaging client** dequeues. This package also contains procedures for creating **queue**s and for managing Streams metadata, such as data dictionary information. This package also contains procedures that enable you to configure and maintain a Streams **replication** environment. This package is provided as an easy way to complete common tasks in a Streams environment. You can use other packages, such as the DBMS_CAPTURE_ADM, DBMS_PROPAGATION_ADM, DBMS_APPLY_ADM, DBMS_RULE_ADM, and DBMS_AQADM packages, to complete these same tasks, as well as tasks that require additional customization.

DBMS_CAPTURE_ADM Package

The DBMS_CAPTURE_ADM package provides an administrative interface for starting, stopping, and configuring a **capture process**. This package also provides administrative procedures that prepare database objects at the **source database** for **instantiation** at a **destination database**.

DBMS_PROPAGATION_ADM Package

The DBMS_PROPAGATION_ADM package provides an administrative interface for configuring propagation from a **source queue** to a **destination queue**.

DBMS_APPLY_ADM Package

The DBMS_APPLY_ADM package provides an administrative interface for starting, stopping, and configuring an apply process. This package includes procedures that enable you to configure **apply handler**s, set enqueue destinations for messages, and specify execution directives for messages. This package also provides administrative procedures that set the **instantiation SCN** for objects at a destination database. This package also includes subprograms for configuring **conflict** detection and resolution and for managing apply errors.

DBMS_STREAMS_MESSAGING Package

The DBMS_STREAMS_MESSAGING package provides interfaces to enqueue messages into and dequeue messages from an ANYDATA queue.

DBMS_RULE_ADM Package

The DBMS_RULE_ADM package provides an administrative interface for creating and managing rules, **rule set**s, and rule **evaluation context**s. This package also contains subprograms for managing privileges related to rules.

DBMS_RULE Package

The DBMS_RULE package contains the EVALUATE procedure, which evaluates a rule set. The goal of this procedure is to produce the list of satisfied rules, based on the data. This package also contains subprograms that enable you to use iterators during rule evaluation. Instead of returning all rules that evaluate to TRUE or MAYBE for an evaluation, iterators can return one rule at a time.

DBMS_STREAMS Package

The DBMS_STREAMS package provides interfaces to convert ANYDATA objects into LCR objects, to return information about Streams attributes and **Streams client**s, and to annotate redo entries generated by a session with a **tag**. This tag can affect the behavior of a capture process, a propagation, an apply process, or a messaging client whose rules include specifications for these tags in redo entries or LCRs.

DBMS_STREAMS_TABLESPACE_ADM

The DBMS_STREAMS_TABLESPACE_ADM package provides administrative procedures for creating and managing a tablespace repository. This package also provides administrative procedures for copying tablespaces between databases and moving tablespaces from one database to another. This package uses transportable tablespaces, Data Pump, and the DBMS_FILE_TRANSFER package.

DBMS_STREAMS_AUTH Package

The DBMS_STREAMS_AUTH package provides interfaces for granting privileges to and revoking privileges from Streams administrators.

Streams Data Dictionary Views

Every database in a Streams environment has Streams data dictionary views. These views maintain administrative information about local **rule**s, objects, **capture process**es, **propagation**s, **apply process**es, and **messaging client**s. You can use these views to monitor your Streams environment.

See Also:

- Chapter 19, "Monitoring a Streams Environment"
- *Oracle Streams Replication Administrator's Guide* for queries that are useful in a Streams **replication** environment
- *Oracle Database Reference* for more information about these data dictionary views

Streams Tool in the Oracle Enterprise Manager Console

To help configure, administer, and monitor Streams environments, Oracle provides a Streams tool in the Oracle Enterprise Manager Console. You can also use the Streams tool to generate Streams configuration scripts, which you can then modify and run to configure your Streams environment. The Streams tool online help contains the primary documentation for this tool.

Figure 1–13 shows the top portion of the Streams page in Enterprise Manager.

Figure 1–13 Streams Page in Enterprise Manager

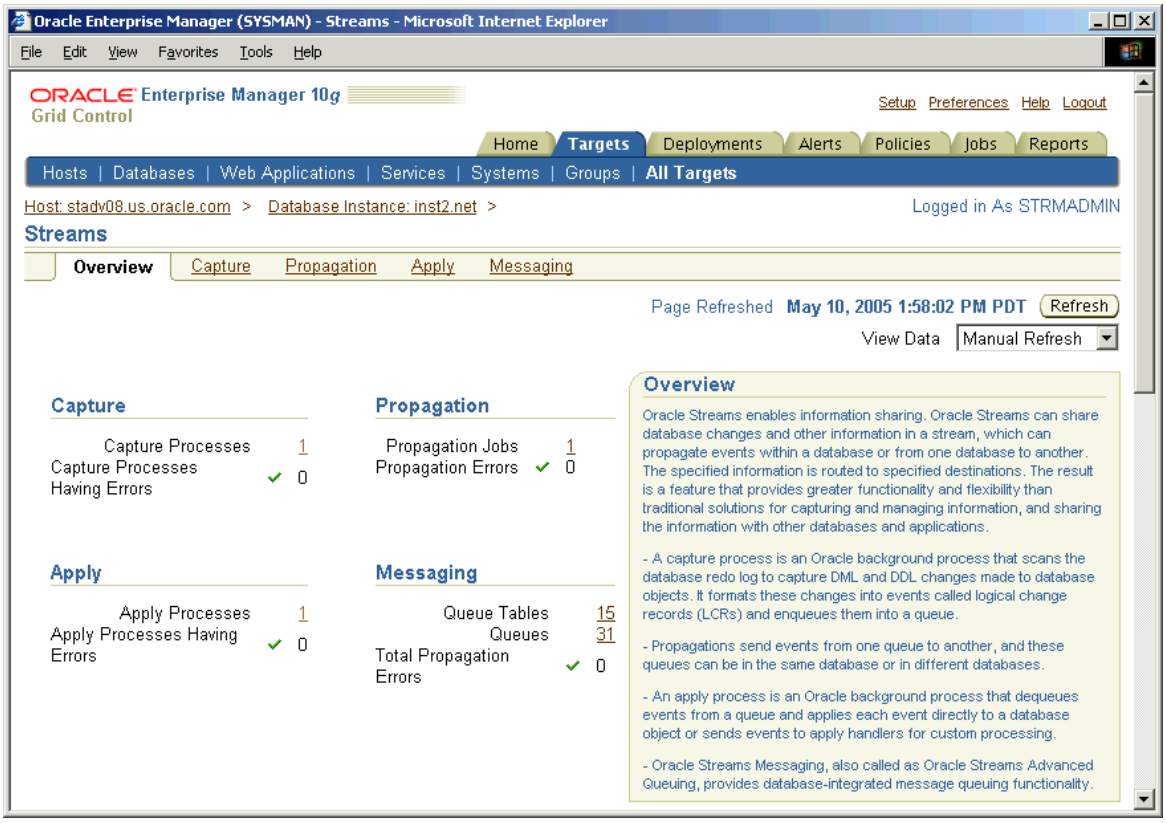
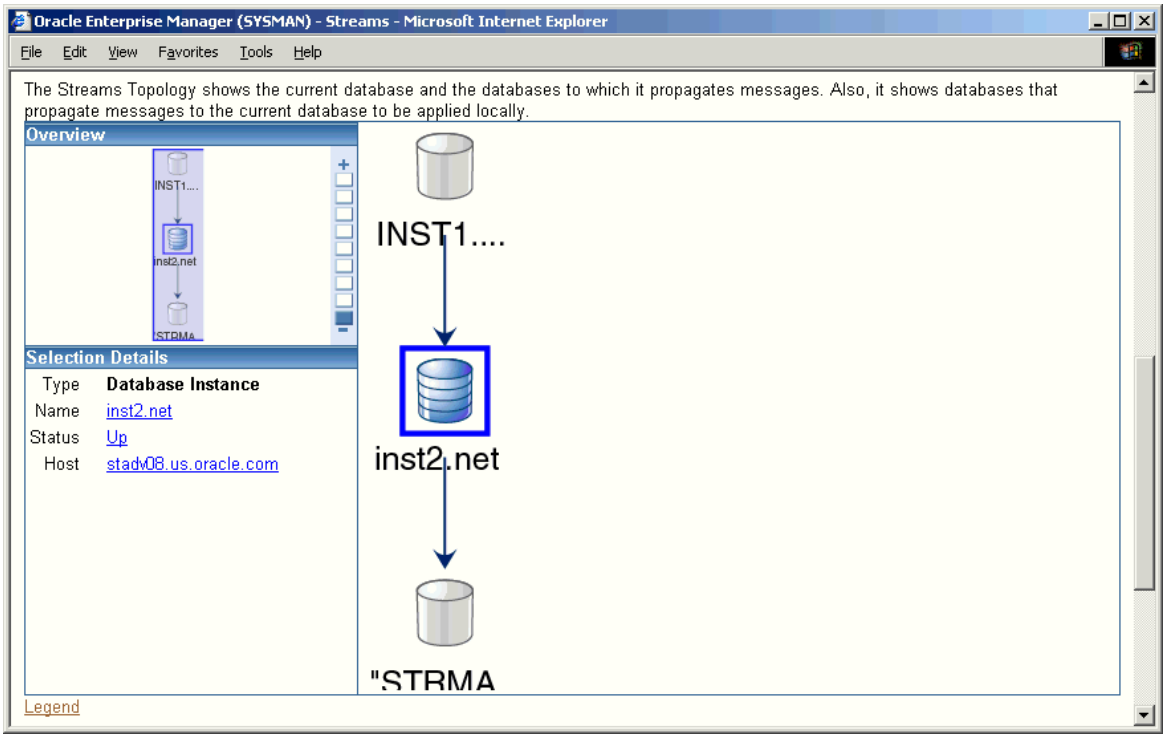


Figure 1–14 shows the Streams Topology, which is on the bottom portion of the Streams page in the Enterprise Manager.

Figure 1–14 Streams Topology



See Also: The online help for the Streams tool in the Oracle Enterprise Manager

2

Streams Capture Process

This chapter explains the concepts and architecture of the Streams **capture process**.

This chapter contains these topics:

See Also: Chapter 11, "Managing a Capture Process"

The Redo Log and a Capture Process

Every Oracle database has a set of two or more redo log files. The redo log files for a database are collectively known as the database redo log. The primary function of the redo log is to record all changes made to the database.

Redo logs are used to guarantee recoverability in the event of human error or media failure. A **capture process** is an optional Oracle background process that scans the database redo log to capture DML and DDL changes made to database objects. When a capture process is configured to capture changes from a redo log, the database where the changes were generated is called the **source database**.

A capture process can run on the source database or on a remote database. When a capture process runs on the source database, the capture process is a **local capture process**. When a capture process runs on a remote database, the capture process is called a **downstream capture process**, and the remote database is called the **downstream database**.

Logical Change Records (LCRs)

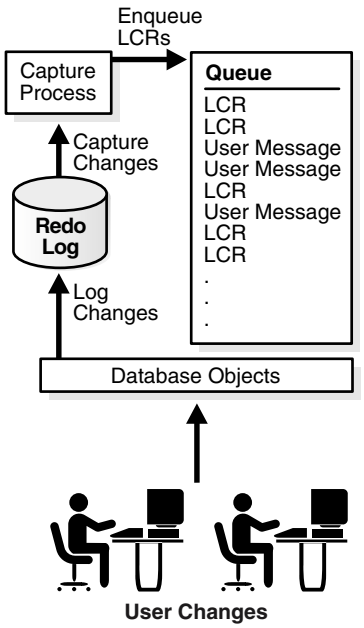
A **capture process** reformats changes captured from the redo log into LCRs. An LCR is a **message** with a specific format that describes a database change. A capture process captures two types of LCRs: **row LCRs** and **DDL LCRs**. Row LCRs and DDL LCRs are described in detail later in this section.

After capturing an LCR, a capture process enqueues a message containing the LCR into a **queue**. A capture process is always associated with a single `ANYDATA` queue, and it enqueues messages into this queue only. For improved performance, **captured message**s always are stored in a **buffered queue**, which is System Global Area (SGA) memory associated with an `ANYDATA` queue. You can create multiple queues and associate a different capture process with each queue.

Figure 2–1 shows a capture process capturing LCRs.

Note: A capture process can be associated only with an `ANYDATA` queue, not with a **typed queue**.

Figure 2–1 Capture Process



See Also:

- *Oracle Streams Replication Administrator's Guide* for information about managing LCRs
- *Oracle Database PL/SQL Packages and Types Reference* for more information about LCR types
- "Buffered Queues" on page 3-21

Row LCRs

A row LCR describes a change to the data in a single row or a change to a single LONG, LONG RAW, or LOB column in a row. The change results from a data manipulation language (DML) statement or a piecewise update to a LOB. For example, a single DML statement can insert or merge multiple rows into a table, can update multiple rows in a table, or can delete multiple rows from a table.

Therefore, a single DML statement can produce multiple row LCRs. That is, a capture process creates an LCR for each row that is changed by the DML statement. In addition, an update to a LONG, LONG RAW, or LOB column in a single row can result in more than one row LCR.

Each row LCR is encapsulated in an object of LCR$_ROW_RECORD type and contains the following attributes:

- source_database_name: The name of the **source database** where the row change occurred.
- command_type: The type of DML statement that produced the change, either INSERT, UPDATE, DELETE, LOB ERASE, LOB WRITE, or LOB TRIM.
- object_owner: The schema name that contains the table with the changed row.
- object_name: The name of the table that contains the changed row.
- tag: A raw **tag** that can be used to track the LCR.
- transaction_id: The identifier of the transaction in which the DML statement was run.
- scn: The system change number (SCN) at the time when the change record was written to the redo log.
- old_values: The old column values related to the change. These are the column values for the row before the DML change. If the type of the DML statement is UPDATE or DELETE, then these old values include some or all of the columns in the changed row before the DML statement. If the type of the DML statement is INSERT, then there are no old values.
- new_values: The new column values related to the change. These are the column values for the row after the DML change. If the type of the DML statement is UPDATE or INSERT, then these new values include some or all of the columns in the changed row after the DML statement. If the type of the DML statement is DELETE, then there are no new values.

A captured row LCR can also contain transaction control statements. These row LCRs contain directives such as COMMIT and ROLLBACK. Such row LCRs are internal and are used by an **apply process** to maintain transaction consistency between a source database and a **destination database**.

DDL LCRs

A DDL LCR describes a data definition language (DDL) change. A DDL statement changes the structure of the database. For example, a DDL statement can create, alter, or drop a database object.

Each DDL LCR contains the following information:

- `source_database_name`: The name of the **source database** where the DDL change occurred.
- `command_type`: The type of DDL statement that produced the change, for example `ALTER TABLE` or `CREATE INDEX`.
- `object_owner`: The schema name of the user who owns the database object on which the DDL statement was run.
- `object_name`: The name of the database object on which the DDL statement was run.
- `object_type`: The type of database object on which the DDL statement was run, for example `TABLE` or `PACKAGE`.
- `ddl_text`: The text of the DDL statement.
- `logon_user`: The logon user, which is the user whose session executed the DDL statement.
- `current_schema`: The schema that is used if no schema is specified for an object in the DDL text.
- `base_table_owner`: The base table owner. If the DDL statement is dependent on a table, then the base table owner is the owner of the table on which it is dependent.
- `base_table_name`: The base table name. If the DDL statement is dependent on a table, then the base table name is the name of the table on which it is dependent.
- `tag`: A raw **tag** that can be used to track the LCR.
- `transaction_id`: The identifier of the transaction in which the DDL statement was run.
- `scn`: The SCN when the change was written to the redo log.

Note: Both row LCRs and DDL LCRs contain the source database name of the database where a change originated. If **captured message**s will be propagated by a **propagation** or applied by an apply process, then, to avoid propagation and apply problems, Oracle recommends that you do not rename the source database after a capture process has started capturing changes.

See Also: The "SQL Command Codes" table in the *Oracle Call Interface Programmer's Guide* for a complete list of the types of DDL statements

Extra Information in LCRs

In addition to the information discussed in the previous sections, row LCRs and DDL LCRs optionally can include the following extra information (or LCR attributes):

- `row_id`: The rowid of the row changed in a row LCR. This attribute is not included in DDL LCRs or row LCRs for index-organized tables.
- `serial#`: The serial number of the session that performed the change captured in the LCR.
- `session#`: The identifier of the session that performed the change captured in the LCR.
- `thread#`: The thread number of the instance in which the change captured in the LCR was performed. Typically, the thread number is relevant only in a Real Application Clusters environment.
- `tx_name`: The name of the transaction that includes the LCR.
- `username`: The name of the current user who performed the change captured in the LCR.

You can use the `INCLUDE_EXTRA_ATTRIBUTE` procedure in the `DBMS_CAPTURE_ADM` package to instruct a capture process to capture one or more extra attributes.

See Also:

- "Managing Extra Attributes in Captured Messages" on page 11-33
- "Viewing the Extra Attributes Captured by Each Capture Process" on page 20-12
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `INCLUDE_EXTRA_ATTRIBUTE` procedure
- *Oracle Database PL/SQL User's Guide and Reference* for more information about the current user

Capture Process Rules

A **capture process** either captures or discards changes based on **rule**s that you define. Each rule specifies the database objects and types of changes for which the rule evaluates to `TRUE`. You can place these rules in a **positive rule set** or **negative rule set** for the capture process.

If a rule evaluates to `TRUE` for a change, and the rule is in the positive rule set for a capture process, then the capture process captures the change. If a rule evaluates to `TRUE` for a change, and the rule is in the negative rule set for a capture process, then the capture process discards the change. If a capture process has both a positive and a negative rule set, then the negative rule set is always evaluated first.

You can specify capture process rules at the following levels:

- A table rule captures or discards either row changes resulting from DML changes or DDL changes to a particular table. Subset rules are table rules that include a subset of the row changes to a particular table.
- A schema rule captures or discards either row changes resulting from DML changes or DDL changes to the database objects in a particular schema.

- A global rule captures or discards either all row changes resulting from DML changes or all DDL changes in the database.

Note: The capture process does not capture certain types of changes and changes to certain datatypes in table columns. Also, a capture process never captures changes in the `SYS`, `SYSTEM`, or `CTXSYS` schemas.

See Also:

- Chapter 5, "Rules"
- Chapter 6, "How Rules Are Used in Streams"

Datatypes Captured

When capturing the row changes resulting from DML changes made to tables, a **capture process** can capture changes made to columns of the following datatypes:

- `VARCHAR2`
- `NVARCHAR2`
- `NUMBER`
- `LONG`
- `DATE`
- `BINARY_FLOAT`
- `BINARY_DOUBLE`
- `TIMESTAMP`
- `TIMESTAMP WITH TIME ZONE`
- `TIMESTAMP WITH LOCAL TIME ZONE`
- `INTERVAL YEAR TO MONTH`
- `INTERVAL DAY TO SECOND`
- `RAW`
- `LONG RAW`
- `CHAR`
- `NCHAR`
- `CLOB`
- `NCLOB`
- `BLOB`
- `UROWID`

A capture process does not capture the results of DML changes to columns of the following datatypes: `BFILE`, `ROWID`, and user-defined types (including object types, `REF`s, varrays, nested tables, and Oracle-supplied types). Also, a capture process cannot capture changes to columns if the columns have been encrypted using transparent data encryption. A capture process raises an error if it tries to create a row

LCR for a DML change to a table containing encrypted columns or a column of an unsupported datatype.

When a capture process raises an error, it writes the LCR that caused the error into its trace file, raises an ORA-00902 error, and becomes disabled. In this case, modify the **rule**s used by the capture process to avoid the error, and restart the capture process.

Note:

- You can add rules to a **negative rule set** for a capture process that instruct the capture process to discard changes to tables with columns of unsupported datatypes. However, if these rules are not simple rules, then a capture process might create a row LCR for the change and continue to process it. In this case, a change that includes an unsupported datatype can cause the capture process to raise an error, even if the change does not satisfy the **rule set**s used by the capture process. The `DBMS_STREAMS_ADM` package creates only simple rules.
- Some of the datatypes listed previously in this section might not be supported by Streams in earlier releases of Oracle. If your Streams environment includes one or more databases from an earlier release of Oracle, then make sure row LCRs do not flow into a database that does not support all of the datatypes in the row LCRs. See the Streams documentation for the earlier Oracle release for information about supported datatypes.

See Also:

- "Simple Rule Conditions" on page 5-3 for information about simple rules
- Chapter 6, "How Rules Are Used in Streams" for more information about rule sets for Streams clients and for information about how messages satisfy rule sets
- "Capture Process Rule Evaluation" on page 2-41
- "Datatypes Applied" on page 4-8 for information about the datatypes that can be applied by an apply process
- *Oracle Database SQL Reference* for more information about datatypes

Types of Changes Captured

A **capture process** can capture only certain types of changes made to a database and its objects. The following sections describe the types of DML and DDL changes that can be captured.

Note: A capture process never captures changes in the `SYS`, `SYSTEM`, or `CTXSYS` schemas.

See Also: Chapter 4, "Streams Apply Process" for information about the types of changes an apply process can apply

Types of DML Changes Captured

When you specify that DML changes made to certain tables should be captured, a capture process captures the following types of DML changes made to these tables:

- `INSERT`
- `UPDATE`
- `DELETE`
- `MERGE`
- Piecewise updates to LOBs

The following are considerations for capturing DML changes:

- A capture process converts each `MERGE` change into an `INSERT` or `UPDATE` change. `MERGE` is not a valid command type in a row LCR.
- A capture process can capture changes made to an index-organized table only if the index-organized table does not contain any columns of the following datatypes:
  - `ROWID`
  - `UROWID`
  - User-defined types (including object types, `REF`s, varrays, and nested tables)

  If an index-organized table contains a column of one of these datatypes, then a capture process raises an error when a user makes a change to the index-organized table and the change satisfies the capture process **rule set**s.
- A capture process ignores `CALL`, `EXPLAIN PLAN`, or `LOCK TABLE` statements.
- A capture process cannot capture DML changes made to temporary tables or object tables. A capture process raises an error if it attempts to capture such changes.
- If you share a sequence at multiple databases, then sequence values used for individual rows at these databases might vary. Also, changes to actual sequence values are not captured. For example, if a user references a `NEXTVAL` or sets the sequence, then a capture process does not capture changes resulting from these operations.

See Also:

- "Datatypes Captured" on page 2-6 for information about the datatypes supported by a capture process
- Chapter 6, "How Rules Are Used in Streams" for more information about rule sets for **Streams client**s and for information about how messages satisfy rule sets
- *Oracle Streams Replication Administrator's Guide* for information about applying DML changes with an **apply process** and for information about strategies to avoid having the same sequence-generated value for two different rows at different databases
- *Oracle XML DB Developer's Guide* for information about SQL functions that update XML data

DDL Changes and Capture Processes

A capture process captures the DDL changes that satisfy its **rule set**s, *except for* the following types of DDL changes:

- `ALTER DATABASE`
- `CREATE CONTROLFILE`
- `CREATE DATABASE`
- `CREATE PFILE`
- `CREATE SPFILE`
- `FLASHBACK DATABASE`

A capture process can capture DDL statements, but not the results of DDL statements, unless the DDL statement is a `CREATE TABLE AS SELECT` statement. For example, when a capture process captures an `ANALYZE` statement, it does not capture the statistics generated by the `ANALYZE` statement. However, when a capture process captures a `CREATE TABLE AS SELECT` statement, it captures the statement itself and all of the rows selected (as `INSERT` row LCRs).

Some types of DDL changes that are captured by a capture process cannot be applied by an **apply process**. If an apply process receives a DDL LCR that specifies an operation that cannot be applied, then the apply process ignores the DDL LCR and records information about it in the trace file for the apply process.

When a capture process captures a DDL change that specifies timestamps or system change number (SCN) values in its syntax, configure a DDL handler for any apply processes that will dequeue the change. The DDL handler must process timestamp or SCN values properly. For example, although a capture process always ignores `FLASHBACK DATABASE` statements, a capture process captures `FLASHBACK TABLE` statements when its rule sets instruct it to capture DDL changes to the specified table. `FLASHBACK TABLE` statements include timestamps or SCN values in its syntax.

See Also:

- *Oracle Streams Replication Administrator's Guide* for information about applying DDL changes with an apply process
- Chapter 6, "How Rules Are Used in Streams" for more information about rule sets for **Streams client**s and for information about how messages satisfy rule sets

Other Types of Changes Ignored by a Capture Process

The following types of changes are ignored by a capture process:

- The session control statements `ALTER SESSION` and `SET ROLE`.
- The system control statement `ALTER SYSTEM`.
- Invocations of PL/SQL procedures, which means that a call to a PL/SQL procedure is not captured. However, if a call to a PL/SQL procedure causes changes to database objects, then these changes can be captured by a capture process if the changes satisfy the capture process **rule set**s.
- Changes made to a table or schema by online redefinition using the `DBMS_REDEFINITION` package. Online table redefinition is supported on a table for which a capture process captures changes, but the logical structure of the table before online redefinition must be the same as the logical structure after online redefinition.

NOLOGGING and UNRECOVERABLE Keywords for SQL Operations

If you use the NOLOGGING or UNRECOVERABLE keyword for a SQL operation, then the changes resulting from the SQL operation cannot be captured by a **capture process**. Therefore, do not use these keywords if you want to capture the changes that result from a SQL operation.

If the object for which you are specifying the logging attributes resides in a database or tablespace in FORCE LOGGING mode, then Oracle ignores any NOLOGGING or UNRECOVERABLE setting until the database or tablespace is taken out of FORCE LOGGING mode. You can determine the current logging mode for a database by querying the FORCE_LOGGING column in the V$DATABASE dynamic performance view. You can determine the current logging mode for a tablespace by querying the FORCE_LOGGING column in the DBA_TABLESPACES static data dictionary view.

Note: The UNRECOVERABLE keyword is deprecated and has been replaced with the NOLOGGING keyword in the *logging_clause*. Although UNRECOVERABLE is supported for backward compatibility, Oracle strongly recommends that you use the NOLOGGING keyword, when appropriate.

See Also: *Oracle Database SQL Reference* for more information about the NOLOGGING and UNRECOVERABLE keywords, FORCE LOGGING mode, and the *logging_clause*

UNRECOVERABLE Clause for Direct Path Loads

If you use the UNRECOVERABLE clause in the SQL*Loader control file for a direct path load, then the changes resulting from the direct path load cannot be captured by a **capture process**. Therefore, if the changes resulting from a direct path load should be captured by a capture process, then do not use the UNRECOVERABLE clause.

If you perform a direct path load without logging changes at a **source database**, but you do not perform a similar direct path load at the **destination database**s of the source database, then apply errors can result at these destination databases when changes are made to the loaded objects at the source database. In this case, a capture process at the source database can capture changes to these objects, and one or more **propagation**s can propagate the changes to the destination databases. When an apply process tries to apply these changes, errors result unless both the changed object and the changed rows in the object exist on the destination database.

Therefore, if you use the UNRECOVERABLE clause for a direct path load and a capture process is configured to capture changes to the loaded objects, then make sure any destination databases contain the loaded objects and the loaded data to avoid apply errors. One way to make sure that these objects exist at the destination databases is to perform a direct path load at each of these destination databases that is similar to the direct path load performed at the source database.

If you load objects into a database or tablespace that is in FORCE LOGGING mode, then Oracle ignores any UNRECOVERABLE clause during a direct path load, and the loaded changes are logged. You can determine the current logging mode for a database by querying the FORCE_LOGGING column in the V$DATABASE dynamic performance view. You can determine the current logging mode for a tablespace by querying the FORCE_LOGGING column in the DBA_TABLESPACES static data dictionary view.

See Also: *Oracle Database Utilities* for information about direct path loads and SQL*Loader

Supplemental Logging in a Streams Environment

Supplemental logging places additional column data into a redo log whenever an operation is performed. A **capture process** captures this additional information and places it in LCRs. Supplemental logging is always configured at a **source database**, regardless of location of the capture process that captures changes to the source database.

Typically, supplemental logging is required in Streams **replication** environments. In these environments, an **apply process** needs the additional information in the LCRs to properly apply DML changes and DDL changes that are replicated from a source database to a **destination database**. However, supplemental logging can also be required in environments where changes are not applied to database objects directly by an apply process. In such environments, an **apply handler** can process the changes without applying them to the database objects, and the supplemental information might be needed by the apply handlers.

See Also: *Oracle Streams Replication Administrator's Guide* for detailed information about when supplemental logging is required

Instantiation in a Streams Environment

In a Streams environment that shares a database object within a single database or between multiple databases, a **source database** is the database where changes to the object are generated in the redo log, and a **destination database** is the database where these changes are dequeued by an **apply process**. If a **capture process** captures or will capture such changes, and the changes will be applied locally or propagated to other databases and applied at destination databases, then you must **instantiate** these source database objects before these changes can be dequeued and processed by an apply process. If a database where changes to the source database objects will be applied is a different database than the source database, then the destination database must have a copy of these database objects.

In Streams, the following general steps instantiate a database object:

1. Prepare the object for instantiation at the source database.
2. If a copy of the object does not exist at the destination database, then create an object physically at the destination database based on an object at the source database. You can use export/import, transportable tablespaces, or RMAN to copy database objects for instantiation. If the database objects already exist at the destination database, then this step is not necessary.
3. Set the **instantiation SCN** for the database object at the destination database. An instantiation SCN instructs an apply process at the destination database to apply only changes that committed at the source database after the specified SCN.

In some cases, Step 1 and Step 3 are completed automatically. For example, when you add **rule**s for an object to the **positive rule set** for a capture process by running a procedure in the `DBMS_STREAMS_ADM` package, the object is prepared for instantiation automatically. Also, when you use export/import or transportable tablespaces to copy database objects from a source database to a destination database, instantiation SCNs can be set for these objects automatically. Instantiation is required whenever an apply process dequeues **captured message**s, even if the apply process sends the LCRs to an **apply handler** that does not execute them.

Note: You can use either Data Pump export/import or original export/import for Streams instantiations. General references to export/import in this document refer to both Data Pump and original export/import. This document distinguishes between Data Pump and original export/import when necessary.

See Also: *Oracle Streams Replication Administrator's Guide* for detailed information about instantiation in a Streams replication environment

Local Capture and Downstream Capture

You can configure a **capture process** to run locally on a **source database** or remotely on a **downstream database**. A single database can have one or more capture processes that capture local changes and other capture processes that capture changes from a remote source database. That is, you can configure a single database to perform both local capture and downstream capture.

Local Capture

Local capture means that a capture process runs on the **source database**. Figure 2–1 on page 2-2 shows a database using local capture.

The Source Database Performs All Change Capture Actions

If you configure local capture, then the following actions are performed at the source database:

- The `DBMS_CAPTURE_ADM.BUILD` procedure is run to extract (or build) the data dictionary to the redo log.
- Supplemental logging at the source database places additional information in the redo log. This information might be needed when captured changes are applied by an **apply process**.
- The first time a capture process is started at the database, Oracle uses the extracted data dictionary information in the redo log to create a **LogMiner data dictionary**, which is separate from the primary data dictionary for the source database. Additional capture processes can use this existing LogMiner data dictionary, or they can create new LogMiner data dictionaries.
- A capture process scans the redo log for changes using LogMiner.
- The **rules engine** evaluates changes based on the **rule**s in one or more of the capture process **rule set**s.
- The capture process enqueues changes that satisfy the rules in its rule sets into a local `ANYDATA` queue.
- If the captured changes are shared with one or more other databases, then one or more **propagation**s propagate these changes from the source database to the other databases.
- If database objects at the source database must be instantiated at a **destination database**, then the objects must be prepared for **instantiation** and a mechanism such as an Export utility must be used to make a copy of the database objects.

Advantages of Local Capture

The following are the advantages of using local capture:

- Configuration and administration of the capture process is simpler than when downstream capture is used. When you use local capture, you do not need to configure redo log file copying to a **downstream database**, and you administer the capture process locally at the database where the captured changes originated.
- A **local capture process** can scan changes in the online redo log before the database writes these changes to an archived redo log file. When you use downstream capture, archived redo log files are copied to the downstream database after the source database has finished writing changes to them, and some time is required to copy the redo log files to the downstream database.
- The amount of data being sent over the network is reduced, because the entire redo log file is not copied to the downstream database. Even if **captured message**s are propagated to other databases, the captured messages can be a subset of the total changes made to the database, and only the LCRs that satisfy the rules in the rule sets for a **propagation** are propagated.
- Security might be improved because only the source (local) database can access the redo log files. For example, if you want to capture changes in the `hr` schema only, then, when you use local capture, only the source database can access the redo log to enqueue changes to the `hr` schema into the capture process **queue**. However, when you use downstream capture, the redo log files are copied to the downstream database, and these redo log files contain all of the changes made to the database, not just the changes made to the `hr` schema.
- Some types of **custom rule-based transformation**s are simpler to configure if the capture process is running at the local source database. For example, if you use local capture, then a custom rule-based transformation can use cached information in a PL/SQL session variable which is populated with data stored at the source database.
- In a Streams environment where messages are captured and applied in the same database, it might be simpler, and use fewer resources, to configure local queries and computations that require information about captured changes and the local data.

Downstream Capture

Downstream capture means that a capture process runs on a database other than the **source database**. The following types of downstream capture configurations are possible: real-time downstream capture and archived-log downstream capture. The `DOWNSTREAM_REAL_TIME_MINE` capture process parameter controls whether a downstream capture process performs real-time downstream capture or archived-log downstream capture. A real-time downstream capture process and one or more archived-log downstream capture processes can coexist at a **downstream database**.

Note:

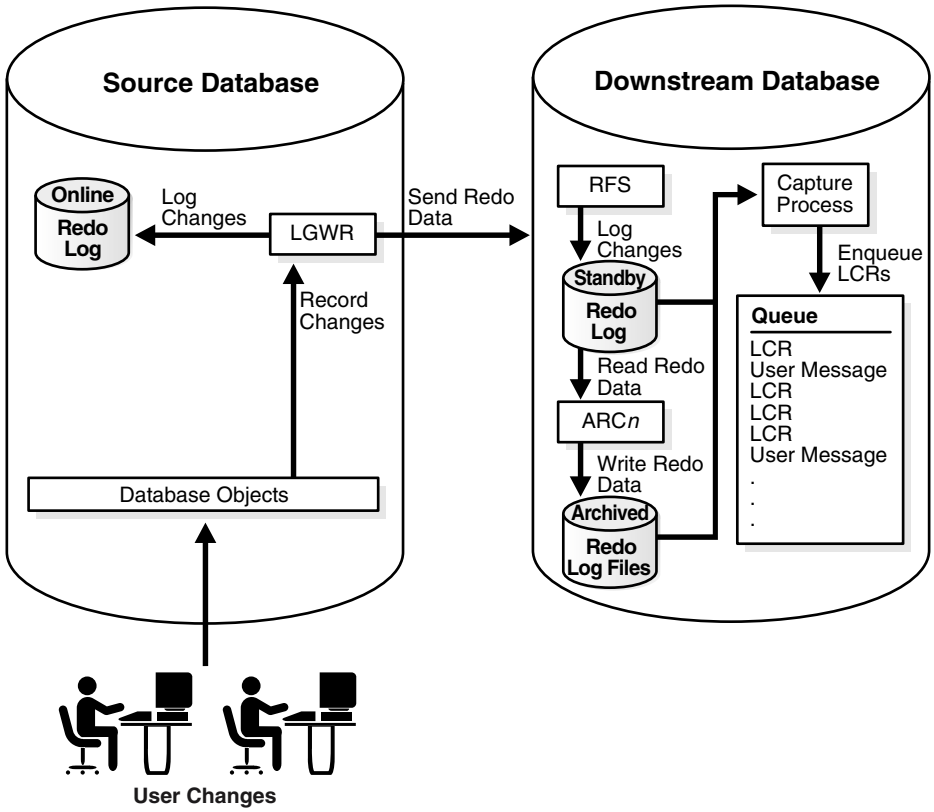
- References to "downstream capture processes" in this document apply to both real-time downstream capture processes and archived-log downstream capture processes. This document distinguishes between the two types of downstream capture processes when necessary.
- A downstream capture process only can capture changes from a single source database. However, multiple downstream capture processes at a single downstream database can capture changes from a single source database or multiple source databases.
- To configure downstream capture, the source database must be an Oracle Database 10*g* Release 1 database or later.

Real-Time Downstream Capture

A **real-time downstream capture** configuration works in the following way:

- Redo transport services use the log writer process (LGWR) at the source database to send redo data to the downstream database either synchronously or asynchronously. At the same time, the LGWR records redo data in the online redo log at the source database.
- A remote file server process (RFS) at the downstream database receives the redo data over the network and stores the redo data in the standby redo log.
- A log switch at the source database causes a log switch at the downstream database, and the `ARCH`*`n`* process at the downstream database archives the current standby redo log file.
- The real-time downstream capture process captures changes from the standby redo log whenever possible and from the archived standby redo log files whenever necessary. A capture process can capture changes in the archived standby redo log files if it falls behind. When it catches up, it resumes capturing changes from the standby redo log.

Figure 2–2 Real-Time Downstream Capture



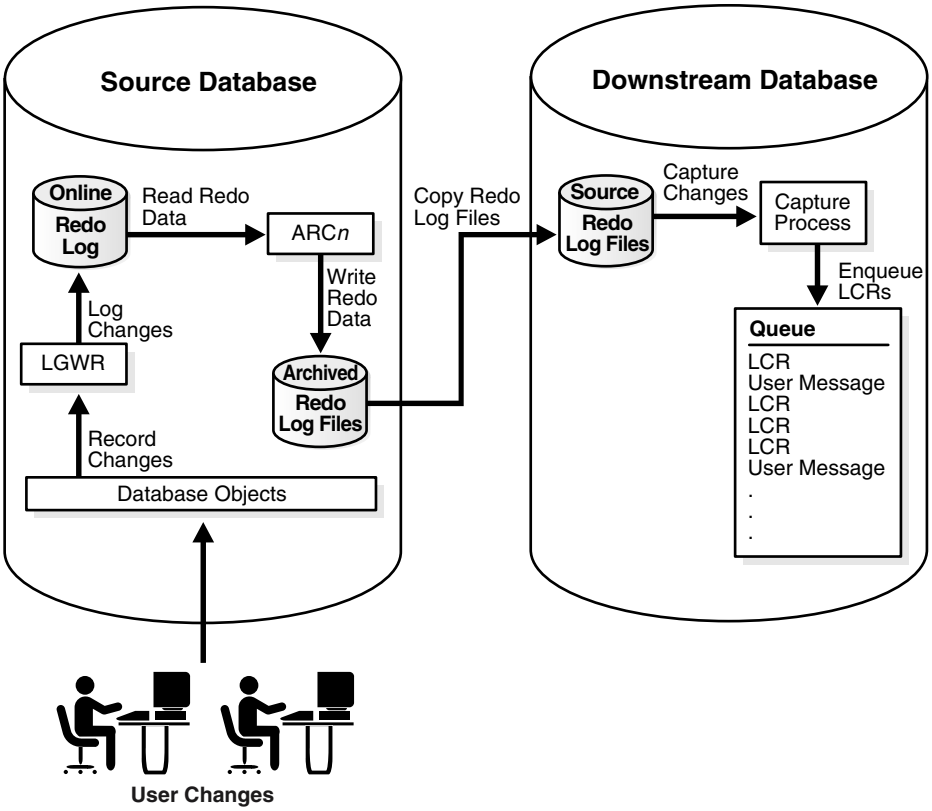
The advantage of real-time downstream capture over archived-log downstream capture is that real-time downstream capture reduces the amount of time required to capture changes made at the source database. The time is reduced because the real-time capture process does not need to wait for the redo log file to be archived before it can capture data from it.

Note: Only one real-time downstream capture process can exist at a downstream database.

Archived-Log Downstream Capture

A **archived-log downstream capture process** configuration means that archived redo log files from the source database are copied to the downstream database, and the capture process captures changes in these archived redo log files. You can copy the archived redo log files to the downstream database using redo transport services, the `DBMS_FILE_TRANSFER` package, file transfer protocol (FTP), or some other mechanism.

Figure 2–3 Archived-Log Downstream Capture



Note: As illustrated in Figure 2–3, the source database for a change captured by a downstream capture process is the database where the change was recorded in the redo log, not the database running the downstream capture process.

The advantage of archived-log downstream capture over real-time downstream capture is that archived-log downstream capture allows multiple downstream capture processes at a downstream database. You can copy redo log files from multiple source databases to a single downstream database and configure multiple archived-log downstream capture processes to capture changes in these redo log files.

See Also: *Oracle Data Guard Concepts and Administration* for more information about redo transport services

The Downstream Database Performs Most Change Capture Actions

If you configure either real-time or archived-log downstream capture, then the following actions are performed at the downstream database:

- The first time a downstream capture process is started at the downstream database, Oracle uses data dictionary information in the redo data from the source database to create a LogMiner data dictionary at the downstream database. The `DBMS_CAPTURE_ADM.BUILD` procedure is run at the source database to extract the source data dictionary information to the redo log at the source database. Next, the redo data is copied to the downstream database from the source database. Additional downstream capture processes for the same source database can use

this existing LogMiner data dictionary, or they can create new LogMiner data dictionaries. Also, a real-time downstream capture process can share a LogMiner data dictionary with one or more archived-log downstream capture processes.

- A capture process scans the redo data from the source database for changes using LogMiner.
- The **rules engine** evaluates changes based on the **rule**s in one or more of the capture process **rule set**s.
- The capture process enqueues changes that satisfy the rules in its rule sets into a local `ANYDATA` queue. The capture process formats the changes as LCRs.
- If the captured messages are shared with one or more other databases, then one or more **propagation**s propagate these LCRs from the downstream database to the other databases.

In a downstream capture configuration, the following actions are performed at the source database:

- The `DBMS_CAPTURE_ADM.BUILD` procedure is run at the source database to extract the data dictionary to the redo log.
- Supplemental logging at the source database places additional information that might be needed for apply in the redo log.
- If database objects at the source database must be instantiated at other databases in the environment, then the objects must be prepared for **instantiation** and a mechanism such as an Export utility must be used to make a copy of the database objects.

In addition, the redo data must be copied from the computer system running the source database to the computer system running the downstream database. In a real-time downstream capture configuration, redo transport services use LWGR to send redo data to the downstream database. Typically, in an archived-log downstream capture configuration, redo transport services copy the archived redo log files to the downstream database.

> **See Also:** Chapter 6, "How Rules Are Used in Streams" for more information about rule sets for **Streams client**s and for information about how messages satisfy rule sets

Advantages of Downstream Capture

The following are the advantages of using downstream capture:

- Capturing changes uses fewer resources at the source database because the downstream database performs most of the required work.
- If you plan to capture changes originating at multiple source databases, then capture process administration can be simplified by running multiple archived-log downstream capture processes with different source databases at one downstream database. That is, one downstream database can act as the central location for change capture from multiple sources. In such a configuration, one real-time downstream capture process can run at the downstream database in addition to the archived-log downstream capture processes.

- Copying redo data to one or more downstream databases provides improved protection against data loss. For example, redo log files at the downstream database can be used for recovery of the source database in some situations.
- The ability to configure at one or more downstream databases multiple capture processes that capture changes from a single source database provides more flexibility and can improve scalability.

Optional Database Link from the Downstream Database to the Source Database

When you create or alter a downstream capture process, you optionally can specify the use of a database link from the downstream database to the source database. This database link must have the same name as the global name of the source database. Such a database link simplifies the creation and administration of a downstream capture process. You specify that a downstream capture process uses a database link by setting the `use_database_link` parameter to `true` when you run `CREATE_CAPTURE` or `ALTER_CAPTURE` on the downstream capture process.

When a downstream capture process uses a database link to the source database, the capture process connects to the source database to perform the following administrative actions automatically:

- In certain situations, runs the `DBMS_CAPTURE_ADM.BUILD` procedure at the source database to extract the data dictionary at the source database to the redo log when a capture process is created.
- Prepares source database objects for instantiation.
- Obtains the **first SCN** for the downstream capture process if the first SCN is not specified during capture process creation. The first SCN is needed to create a capture process.

If a downstream capture process does not use a database link, then you must perform these actions manually.

> **See Also:** "Preparing for and Creating a Real-Time Downstream Capture Process" on page 11-6 for information about when the `DBMS_CAPTURE_ADM.BUILD` procedure is run automatically during capture process creation if the downstream capture process uses a database link

Operational Requirements for Downstream Capture

The following are operational requirements for using downstream capture:

- The source database must be running at least Oracle Database 10*g* and the downstream capture database must be running the same release of Oracle as the source database or later.
- The downstream database must be running Oracle Database 10*g* Release 2 to configure real-time downstream capture. In this case, the source database must be running Oracle Database 10*g* Release 1 or later.
- The operating system on the source and downstream capture sites must be the same, but the operating system release does not need to be the same. In addition, the downstream sites can use a different directory structure from the source site.

- The hardware architecture on the source and downstream capture sites must be the same. For example, a downstream capture configuration with a source database on a 32-bit Sun system must have a downstream database that is configured on a 32-bit Sun system. Other hardware elements, such as the number of CPUs, memory size, and storage configuration, can be different between the source and downstream sites.

In a downstream capture environment, the source database can be a single instance database or a multi-instance Real Application Clusters (RAC) database. The downstream database can be a single instance database or a multi-instance RAC database, regardless of whether the source database is single instance or multi-instance.

SCN Values Relating to a Capture Process

This section describes system change number (SCN) values that are important for a **capture process**. You can query the DBA_CAPTURE data dictionary view to display these values for one or more capture processes.

Captured SCN and Applied SCN

The **captured SCN** is the SCN that corresponds to the most recent change scanned in the redo log by a capture process. The **applied SCN** for a capture process is the SCN of the most recent **message** dequeued by the relevant **apply process**es. All messages lower than this SCN have been dequeued by all apply processes that apply changes captured by the capture process. The applied SCN for a capture process is equivalent to the **low-watermark** SCN for an apply process that applies changes captured by the capture process.

First SCN and Start SCN

This section describes the first SCN and start SCN for a capture process.

First SCN

The **first SCN** is the lowest SCN in the redo log from which a capture process can capture changes. If you specify a first SCN during capture process creation, then the database must be able to access redo data from the SCN specified and higher.

The DBMS_CAPTURE_ADM.BUILD procedure extracts the **source database** data dictionary to the redo log. When you create a capture process, you can specify a first SCN that corresponds to this data dictionary build in the redo log. Specifically, the first SCN for the capture process being created can be set to any value returned by the following query:

```
COLUMN FIRST_CHANGE# HEADING 'First SCN' FORMAT 999999999
COLUMN NAME HEADING 'Log File Name' FORMAT A50

SELECT DISTINCT FIRST_CHANGE#, NAME FROM V$ARCHIVED_LOG
  WHERE DICTIONARY_BEGIN = 'YES';
```

The value returned for the NAME column is the name of the redo log file that contains the SCN corresponding to the first SCN. This redo log file, and subsequent redo log files, must be available to the capture process. If this query returns multiple distinct

values for FIRST_CHANGE#, then the DBMS_CAPTURE_ADM.BUILD procedure has been run more than once on the source database. In this case, choose the first SCN value that is most appropriate for the capture process you are creating.

In some cases, the DBMS_CAPTURE_ADM.BUILD procedure is run automatically when a capture process is created. When this happens, the first SCN for the capture process corresponds to this data dictionary build.

Start SCN

The **start SCN** is the SCN from which a capture process begins to capture changes. You can specify a start SCN that is different than the first SCN during capture process creation, or you can alter a capture process to set its start SCN. The start SCN does not need to be modified for normal operation of a capture process. Typically, you reset the start SCN for a capture process if point-in-time recovery must be performed on one of the **destination database**s that receive changes from the capture process. In these cases, the capture process can be used to capture the changes made at the source database after the point-in-time of the recovery.

Start SCN Must Be Greater than or Equal to First SCN

If you specify a start SCN when you create or alter a capture process, then the start SCN specified must be greater than or equal to the first SCN for the capture process. A capture process always scans any unscanned redo log records that have higher SCN values than the first SCN, even if the redo log records have lower SCN values than the start SCN. So, if you specify a start SCN that is greater than the first SCN, then the capture process might scan redo log records for which it cannot capture changes, because these redo log records have a lower SCN than the start SCN.

Scanning redo log records before the start SCN should be avoided if possible because it can take some time. Therefore, Oracle recommends that the difference between the first SCN and start SCN be as small as possible during capture process creation to keep the initial capture process startup time to a minimum.

Attention: When a capture process is started or restarted, it might need to scan redo log files with a FIRST_CHANGE# value that is lower than start SCN. Removing required redo log files before they are scanned by a capture process causes the capture process to abort. You can query the DBA_CAPTURE data dictionary view to determine the first SCN, start SCN, and **required checkpoint SCN**. A capture process needs the redo log file that includes the required checkpoint SCN, and all subsequent redo log files.

See Also: "Capture Process Creation" on page 2-27 for more information about the first SCN and start SCN for a capture process

A Start SCN Setting that Is Prior to Preparation for Instantiation

If you want to capture changes to a database object and apply these changes using an **apply process**, then only changes that occurred after the database object has been prepared for **instantiation** can be applied. Therefore, if you set the start SCN for a capture process lower than the SCN that corresponds to the time when a database object was prepared for instantiation, then any captured changes to this database object prior to the prepare SCN cannot be applied by an apply process.

This limitation can be important during capture process creation. If a database object was never prepared for instantiation prior to the time of capture process creation, then

an apply process cannot apply any captured changes to the object from a time before capture process creation time.

In some cases, database objects might have been prepared for instantiation before a new capture process is created. For example, if you want to create a new capture process for a source database whose changes are already being captured by one or more existing capture processes, then some or all of the database objects might have been prepared for instantiation before the new capture process is created. If you want to capture changes to a certain database object with a new capture process from a time before the new capture process was created, then the following conditions must be met for an apply process to apply these captured changes:

- The database object must have been prepared for instantiation before the new capture process is created.
- The start SCN for the new capture process must correspond to a time before the database object was prepared for instantiation.
- The redo logs for the time corresponding to the specified start SCN must be available. Additional redo logs previous to the start SCN might be required as well.

See Also:

- *Oracle Streams Replication Administrator's Guide* for more information about preparing database objects for instantiation
- "Capture Process Creation" on page 2-27

Streams Capture Processes and RESTRICTED SESSION

When you enable restricted session during system startup by issuing a `STARTUP RESTRICT` statement, **capture process**es do not start, even if they were running when the database shut down. When restricted session is disabled with an `ALTER SYSTEM` statement, each capture process that was running when the database shut down is started.

When restricted session is enabled in a running database by the SQL statement `ALTER SYSTEM ENABLE RESTRICTED SESSION` clause, it does not affect any running capture processes. These capture processes continue to run and capture changes. If a stopped capture process is started in a restricted session, then the capture process does not actually start until the restricted session is disabled.

Streams Capture Processes and Oracle Real Application Clusters

You can configure a Streams **capture process** to capture changes in an Oracle Real Application Clusters (RAC) environment. If you use one or more capture processes and RAC in the same environment, then all archived logs that contain changes to be captured by a capture process must be available for all instances in the RAC environment. In a RAC environment, a capture process reads changes made by all instances.

Each capture process is started and stopped on the owner instance for its `ANYDATA` queue, even if the start or stop procedure is run on a different instance. Also, a capture process will follow its **queue** to a different instance if the current owner instance becomes unavailable. The queue itself follows the rules for primary instance and secondary instance ownership. If the owner instance for a **queue table** containing a queue used by a capture process becomes unavailable, then queue ownership is transferred automatically to another instance in the cluster. In addition, if the capture

process was enabled when the owner instance became unavailable, then the capture process is restarted automatically on the new owner instance. If the capture process was disabled when the owner instance became unavailable, then the capture process remains disabled on the new owner instance.

The DBA_QUEUE_TABLES data dictionary view contains information about the owner instance for a queue table. Also, any parallel execution servers used by a single capture process run on a single instance in a RAC environment.

LogMiner supports the LOG_ARCHIVE_DEST_*n* initialization parameter, and Streams capture processes use LogMiner to capture changes from the redo log. If an archived log file is inaccessible from one destination, a **local capture process** can read it from another accessible destination. On a RAC database, this ability also enables you to use cross instance archival (CIA) such that each instance archives its files to all other instances. This solution cannot detect or resolve gaps caused by missing archived log files. Hence, it can be used only to complement an existing solution to have the archived files shared between all instances.

See Also:

- "Queues and Oracle Real Application Clusters" on page 3-13 for information about primary and secondary instance ownership for queues
- "Streams Apply Processes and Oracle Real Application Clusters" on page 4-9
- *Oracle Database Reference* for more information about the DBA_QUEUE_TABLES data dictionary view
- *Oracle Database Oracle Clusterware and Oracle Real Application Clusters Administration and Deployment Guide* for more information about configuring archived logs to be shared between instances

Capture Process Architecture

A **capture process** is an optional Oracle background process whose process name is c*nnn*, where *nnn* is a capture process number. Valid capture process names include c001 through c999. A capture process captures changes from the redo log by using the infrastructure of LogMiner. Streams configures LogMiner automatically. You can create, alter, start, stop, and drop a capture process, and you can define capture process **rule**s that control which changes a capture process captures.

Changes are captured in the security domain of the **capture user** for a capture process. The capture user captures all changes that satisfy the capture process **rule set**s. In addition, the capture user runs all **custom rule-based transformation**s specified by the rules in these rule sets. The capture user must have the necessary privileges to perform these actions, including EXECUTE privilege on the rule sets used by the capture process, EXECUTE privilege on all custom rule-based transformation functions specified for rules in the **positive rule set**, and privileges to enqueue **message**s into the capture process **queue**. A capture process can be associated with only one user, but one user can be associated with many capture processes.

See Also: "Configuring a Streams Administrator" on page 10-1 for information about the required privileges

This section discusses the following topics:

Capture Process Components

A capture process consists of the following components:

- One **reader server** that reads the redo log and divides the redo log into regions.
- One or more **preparer servers** that scan the regions defined by the reader server in parallel and perform prefiltering of changes found in the redo log. Prefiltering involves sending partial information about changes, such as schema and object name for a change, to the **rules engine** for evaluation, and receiving the results of the evaluation.
- One **builder server** that merges redo records from the preparer servers. These redo records either evaluated to `TRUE` during partial evaluation or partial evaluation was inconclusive for them. The builder server preserves the SCN order of these redo records and passes the merged redo records to the capture process.
- The capture process (`c`*nnn*) performs the following actions for each change when it receives merged redo records from the builder server:
  - Formats the change into an LCR
  - If the partial evaluation performed by a preparer server was inconclusive for the change in the LCR, then sends the LCR to the **rules engine** for full evaluation
  - Receives the results of the full evaluation of the LCR if it was performed
  - Enqueues the LCR into the **queue** associated with the capture process if the LCR satisfies the **rule**s in the **positive rule set** for the capture process, or discards the LCR if it satisfies the rules in the **negative rule set** for the capture process or if it does not satisfy the rules in the positive rule set

Each reader server, preparer server, and builder server is a parallel execution server. A capture process (`c`*nnn*) is an Oracle background process.

See Also:

- "Capture Process Parallelism" on page 2-40 for more information about the `parallelism` parameter
- "Capture Process Rule Evaluation" on page 2-41
- *Oracle Database Administrator's Guide* for information about managing parallel execution servers

Capture Process States

The state of a capture process describes what the capture process is doing currently. You can view the state of a capture process by querying the `STATE` column in the `V$STREAMS_CAPTURE` dynamic performance view. The following capture process states are possible:

- `INITIALIZING` - Starting up.
- `WAITING FOR DICTIONARY REDO` - Waiting for redo log files containing the dictionary build related to the **first SCN** to be added to the capture process session. A capture process cannot begin to scan the redo log files until all of the log files containing the dictionary build have been added.
- `DICTIONARY INITIALIZATION` - Processing a dictionary build.
- `MINING (PROCESSED SCN = scn_value)` - Mining a dictionary build at the SCN *scn_value*.
- `LOADING (step X of Y)` - Processing information from a dictionary build and currently at step *X* in a process that involves *Y* steps, where *X* and *Y* are numbers.
- `CAPTURING CHANGES` - Scanning the redo log for changes that evaluate to `TRUE` against the capture process **rule set**s.
- `WAITING FOR REDO` - Waiting for new redo log files to be added to the capture process session. The capture process has finished processing all of the redo log files added to its session. This state is possible if there is no activity at a **source database**. For a **downstream capture process**, this state is possible if the capture process is waiting for new log files to be added to its session.
- `EVALUATING RULE` - Evaluating a change against a capture process rule set.
- `CREATING LCR` - Converting a change into an LCR.
- `ENQUEUING MESSAGE` - Enqueuing an LCR that satisfies the capture process rule sets into the capture process **queue**.
- `PAUSED FOR FLOW CONTROL` - Unable to enqueue LCRs either because of low memory or because **propagation**s and apply processes are consuming **message**s slower than the capture process is creating them. This state indicates flow control that is used to reduce spilling of **captured message**s when propagation or apply has fallen behind or is unavailable.
- `SHUTTING DOWN` - Stopping.

See Also: "Displaying Change Capture Information About Each Capture Process" on page 20-3 for a query that displays the state of a capture process

Multiple Capture Processes in a Single Database

If you run multiple capture processes in a single database, consider increasing the size of the System Global Area (SGA) for each instance. Use the SGA_MAX_SIZE initialization parameter to increase the SGA size. Also, if the size of the **Streams pool** is not managed automatically in the database, then you should increase the size of the Streams pool by 10 MB for each capture process parallelism. For example, if you have two capture processes running in a database, and the parallelism parameter is set to 4 for one of them and 1 for the other, then increase the Streams pool by 50 MB (4 + 1 = 5 parallelism).

Also, Oracle recommends that each ANYDATA queue used by a **capture process**, **propagation**, or **apply process** have **captured message**s from at most one capture process from a particular **source database**. Therefore, a separate **queue** should be used for each capture process that captures changes originating at a particular source database.

Note: The size of the Streams pool is managed automatically if the SGA_TARGET initialization parameter is set to a nonzero value.

See Also:

- "Streams Pool" on page 3-19
- "Setting Initialization Parameters Relevant to Streams" on page 10-4 for more information about the STREAMS_POOL_SIZE initialization parameter

Capture Process Checkpoints

A **checkpoint** is information about the current state of a capture process that is stored persistently in the data dictionary of the database running the capture process. A capture process tries to record a checkpoint at regular intervals called **checkpoint intervals**.

Required Checkpoint SCN

The SCN that corresponds to the lowest checkpoint for which a capture process requires redo data is the **required checkpoint SCN**. The redo log file that contains the required checkpoint SCN, and all subsequent redo log files, must be available to the capture process. If a capture process is stopped and restarted, then it starts scanning the redo log from the SCN that corresponds to its required checkpoint SCN. The required checkpoint SCN is important for recovery if a database stops unexpectedly. Also, if the **first SCN** is reset for a capture process, then it must be set to a value that is less than or equal to the required checkpoint SCN for the captured process. You can determine the required checkpoint SCN for a capture process by querying the REQUIRED_CHECKPOINT_SCN column in the DBA_CAPTURE data dictionary view.

See Also: "Displaying the Redo Log Files that Are Required by Each Capture Process" on page 20-8

Maximum Checkpoint SCN

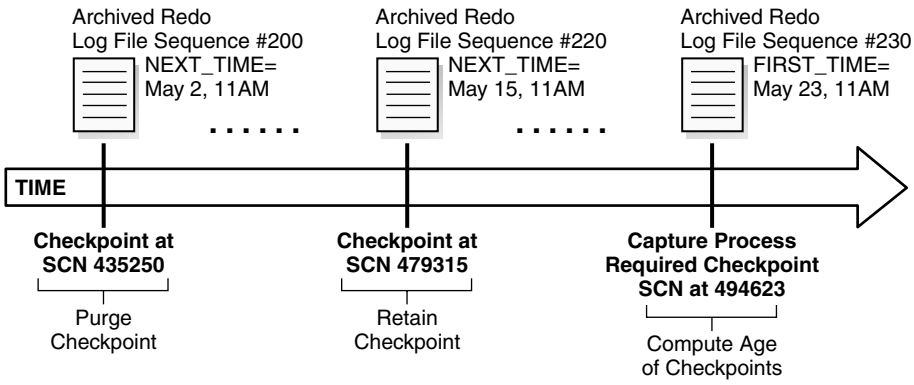
The SCN that corresponds to the last checkpoint recorded by a capture process is the **maximum checkpoint SCN**. If you create a capture process that captures changes from a **source database**, and other capture processes already exist which capture changes from the same source database, then the maximum checkpoint SCNs of the

existing capture processes can help you decide whether the new capture process should create a new **LogMiner data dictionary** or share one of the existing LogMiner data dictionaries. You can determine the maximum checkpoint SCN for a capture process by querying the MAX_CHECKPOINT_SCN column in the DBA_CAPTURE data dictionary view.

Checkpoint Retention Time

The **checkpoint retention time** is the amount of time, in number of days, that a capture process retains checkpoints before purging them automatically. A capture process periodically computes the age of a checkpoint by subtracting the NEXT_TIME of the archived redo log that corresponds to the checkpoint from FIRST_TIME of the archived redo log file containing the required checkpoint SCN for the capture process. If the resulting value is greater than the checkpoint retention time, then the capture process automatically purges the checkpoint by advancing its **first SCN** value. Otherwise, the checkpoint is retained. The DBA_REGISTERED_ARCHIVED_LOG view displays the FIRST_TIME and NEXT_TIME for archived redo log files, and the REQUIRED_CHECKPOINT_SCN column in the DBA_CAPTURE view displays the required checkpoint SCN for a capture process. Figure 2–4 shows an example of a checkpoint being purged when the checkpoint retention time is set to 20 days.

Figure 2–4 Checkpoint Retention Time Set to 20 Days



In Figure 2–4, with the checkpoint retention time set to 20 days, the checkpoint at SCN 435250 is purged because it is 21 days old, while the checkpoint at SCN 479315 is retained because it is 8 days old.

Whenever the first SCN is reset for a capture process, the capture process purges information about archived redo log files prior to the new first SCN from its LogMiner data dictionary. After this information is purged, the archived redo log files remain on the hard disk, but the files are not needed by the capture process. The PURGEABLE column in the DBA_REGISTERED_ARCHIVED_LOG view displays YES for the archived redo log files that are no longer needed. These files can be removed from disk or moved to another location without affecting the capture process.

If you create a capture process using the CREATE_CAPTURE procedure in the DBMS_CAPTURE_ADM package, then you can specify the checkpoint retention time, in days, using the checkpoint_retention_time parameter. The default checkpoint retention time is 60 days if the checkpoint_retention_time parameter is not specified in the CREATE_CAPTURE procedure, or if you use the DBMS_STREAMS_ADM package to create the capture process. The CHECKPOINT_RETENTION_TIME column in the DBA_CAPTURE view displays the current checkpoint retention time for a capture process.

You can change the checkpoint retention time for a capture process by specifying a new time in the ALTER_CAPTURE procedure in the DBMS_CAPTURE_ADM package. If you do not want checkpoints for a capture process to be purged automatically, then specify DBMS_CAPTURE_ADM.INFINITE for the checkpoint_retention_time parameter in CREATE_CAPTURE or ALTER_CAPTURE.

Note: To specify a checkpoint retention time for a capture process, the compatibility level of the database running the capture process must be 10.2.0 or higher. If the compatibility level is lower than 10.2.0 for a database, then the checkpoint retention time for all capture processes running on the database is infinite.

See Also:

- "The LogMiner Data Dictionary for a Capture Process" on page 2-28
- "First SCN and Start SCN Specifications During Capture Process Creation" on page 2-33
- "A New First SCN Value and Purged LogMiner Data Dictionary Information" on page 2-36
- "Managing the Checkpoint Retention Time for a Capture Process" on page 11-29
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the CREATE_CAPTURE and ALTER_CAPTURE procedures

Capture Process Creation

You can create a capture process using the DBMS_STREAMS_ADM package or the DBMS_CAPTURE_ADM package. Using the DBMS_STREAMS_ADM package to create a capture process is simpler because defaults are used automatically for some configuration options. In addition, when you use the DBMS_STREAMS_ADM package, a **rule set** is created for the capture process and **rule**s can be added to the rule set automatically. The rule set is a **positive rule set** if the inclusion_rule parameter is set to true (the default), or it is a **negative rule set** if the inclusion_rule parameter is set to false.

Alternatively, using the DBMS_CAPTURE_ADM package to create a capture process is more flexible, and you create one or more rule sets and rules for the capture process either before or after it is created. You can use the procedures in the DBMS_STREAMS_ADM package or the DBMS_RULE_ADM package to add rules to a rule set for the capture process. To create a capture process at a **downstream database**, you must use the DBMS_CAPTURE_ADM package.

When you create a capture process using a procedure in the DBMS_STREAMS_ADM package and generate one or more rules in the positive rule set for the capture process, the objects for which changes are captured are prepared for **instantiation** automatically, unless it is a **downstream capture process** and there is no database link from the downstream database to the **source database**.

When you create a capture process using the CREATE_CAPTURE procedure in the DBMS_CAPTURE_ADM package, you should prepare for instantiation any objects for which you plan to capture changes as soon as possible after capture process creation.

You can prepare objects for instantiation using one of the following procedures in the DBMS_CAPTURE_ADM package:

- PREPARE_TABLE_INSTANTIATION prepares a single table for instantiation.
- PREPARE_SCHEMA_INSTANTIATION prepares for instantiation all of the objects in a schema and all objects added to the schema in the future.
- PREPARE_GLOBAL_INSTANTIATION prepares for instantiation all of the objects in a database and all objects added to the database in the future.

These procedures can also enable **supplemental logging** for the key columns or for all columns in the table or tables prepared for instantiation.

Note: After creating a capture process, avoid changing the DBID or global name of the source database for the capture process. If you change either the DBID or global name of the source database, then the capture process must be dropped and re-created.

See Also:

- Chapter 11, "Managing a Capture Process" and *Oracle Database PL/SQL Packages and Types Reference* for more information about the following procedures, which can be used to create a capture process:

  DBMS_STREAMS_ADM.ADD_SUBSET_RULES

  DBMS_STREAMS_ADM.ADD_TABLE_RULES

  DBMS_STREAMS_ADM.ADD_SCHEMA_RULES

  DBMS_STREAMS_ADM.ADD_GLOBAL_RULES

  DBMS_CAPTURE_ADM.CREATE_CAPTURE

- *Oracle Streams Replication Administrator's Guide* for more information about capture process rules and preparation for instantiation, and for more information about changing the DBID or global name of a source database

The LogMiner Data Dictionary for a Capture Process

A capture process requires a data dictionary that is separate from the primary data dictionary for the source database. This separate data dictionary is called a **LogMiner data dictionary**. There can be more than one LogMiner data dictionary for a particular source database. If there are multiple capture processes capturing changes from the source database, then two or more capture processes can share a LogMiner data dictionary, or each capture process can have its own LogMiner data dictionary. If the LogMiner data dictionary needed by a capture process does not exist, then the capture process populates it using information in the redo log when the capture process is started for the first time.

The DBMS_CAPTURE_ADM.BUILD procedure extracts data dictionary information to the redo log, and this procedure must be run at least once on the source database before any capture process capturing changes originating at the source database is started. The extracted data dictionary information in the redo log is consistent with the primary data dictionary at the time when the DBMS_CAPTURE_ADM.BUILD procedure is run. This procedure also identifies a valid **first SCN** value that can be used to create a capture process.

You can perform a build of data dictionary information in the redo log multiple times, and a particular build might or might not be used by a capture process to create a LogMiner data dictionary. The amount of information extracted to a redo log when you run the BUILD procedure depends on the number of database objects in the database. Typically, the BUILD procedure generates a large amount of redo data that a capture process must scan subsequently. Therefore, you should run the BUILD procedure only when necessary.

In most cases, if a build is required when a capture process is created using a procedure in the DBMS_STREAMS_ADM or DBMS_CAPTURE_ADM package, then the procedure runs the BUILD procedure automatically. However, the BUILD procedure is not run automatically during capture process creation in the following cases:

- You use CREATE_CAPTURE and specify a non-NULL value for the first_scn parameter. In this case, the specified first SCN must correspond to a previous build.
- You create a downstream capture process that does not use a database link. In this case, the command at the downstream database cannot communicate with the source database to run the BUILD procedure automatically. Therefore, you must run it manually on the source database and specify the first SCN that corresponds to the build during capture process creation.

A capture process requires a LogMiner data dictionary because the information in the primary data dictionary might not apply to the changes being captured from the redo log. These changes might have occurred minutes, hours, or even days before they are captured by a capture process. For example, consider the following scenario:

1. A capture process is configured to capture changes to tables.
2. A database administrator stops the capture process. When the capture process is stopped, it records the SCN of the change it was currently capturing.
3. User applications continue to make changes to the tables while the capture process is stopped.
4. The capture process is restarted three hours after it was stopped.

In this case, to ensure data consistency, the capture process must begin capturing changes in the redo log at the time when it was stopped. The capture process starts capturing changes at the SCN that it recorded when it was stopped.

The redo log contains raw data. It does not contain database object names and column names in tables. Instead, it uses object numbers and internal column numbers for database objects and columns, respectively. Therefore, when a change is captured, a capture process must reference a data dictionary to determine the details of the change.

Because a LogMiner data dictionary might be populated when a capture process is started for the first time, it might take some time to start capturing changes. The amount of time required depends on the number of database objects in the database. You can query the STATE column in the V$STREAMS_CAPTURE dynamic performance view to monitor the progress while a capture process is processing a data dictionary build.

See Also:

- "Capture Process Rule Evaluation" on page 2-41
- "First SCN and Start SCN" on page 2-19
- "Capture Process States" on page 2-24
- *Oracle Streams Replication Administrator's Guide* for more information about preparing database objects for instantiation

Scenario Illustrating Why a Capture Process Needs a LogMiner Data Dictionary Consider a scenario in which a capture process has been configured to capture changes to table `t1`, which has columns `a` and `b`, and the following changes are made to this table at three different points in time:

Time 1: Insert values `a=7` and `b=15`.

Time 2: Add column `c`.

Time 3: Drop column `b`.

If for some reason the capture process is capturing changes from an earlier time, then the primary data dictionary and the relevant version in the LogMiner data dictionary contain different information. Table 2–1 illustrates how the information in the LogMiner data dictionary is used when the current time is different than the change capturing time.

Table 2–1 Information About Table t1 in the Primary and LogMiner Data Dictionaries

Current Time	Change Capturing Time	Primary Data Dictionary	LogMiner Data Dictionary
1	1	Table `t1` has columns `a` and `b`.	Table `t1` has columns `a` and `b` at time 1.
2	1	Table `t1` has columns `a`, `b`, and `c`.	Table `t1` has columns `a` and `b` at time 1.
3	1	Table `t1` has columns `a` and `c`.	Table `t1` has columns `a` and `b` at time 1.

Assume that the capture process captures the change resulting from the insert at time 1 when the actual time is time 3. If the capture process used the primary data dictionary, then it might assume that a value of `7` was inserted into column `a` and a value of `15` was inserted into column `c`, because those are the two columns for table `t1` at time 3 in the primary data dictionary. However, a value of `15` actually was inserted into column `b`, not column `c`.

Because the capture process uses the LogMiner data dictionary, the error is avoided. The LogMiner data dictionary is synchronized with the capture process and continues to record that table `t1` has columns `a` and `b` at time 1. So, the captured change specifies that a value of `15` was inserted into column `b`.

Multiple Capture Processes for the Same Source Database If one or more capture processes are capturing changes made to a source database, and you want to create a new capture process that captures changes to the same source database, then the new capture process can either create a new LogMiner data dictionary or share one of the existing LogMiner data dictionaries with one or more other capture processes. Whether a new LogMiner data dictionary is created for a new capture process depends on the setting for the `first_scn` parameter when you run `CREATE_CAPTURE` to create a capture process:

- If you specify NULL for the first_scn parameter, then the new capture process attempts to share a LogMiner data dictionary with one or more existing capture processes that capture changes from the same source database. NULL is the default for the first_scn parameter.
- If you specify a non-NULL value for the first_scn parameter, then the new capture process uses a new LogMiner data dictionary that is created when the new capture process is started for the first time.

Note:

- When you create a capture process and specify a non-NULL first_scn parameter value, this value should correspond to a data dictionary build in the redo log obtained by running the DBMS_CAPTURE_ADM.BUILD procedure.
- During capture process creation, if the first_scn parameter is NULL and the start_scn parameter is non-NULL, then an error is raised if the start_scn parameter setting is lower than all of the first SCN values for all existing capture processes.

If multiple LogMiner data dictionaries exist, and you specify NULL for the first_scn parameter during capture process creation, then the new capture process automatically attempts to share the LogMiner data dictionary of one of the existing capture processes that has taken at least one **checkpoint**. You can view the **maximum checkpoint SCN** for all existing capture processes by querying the MAX_CHECKPOINT_SCN column in the DBA_CAPTURE data dictionary view.

If multiple LogMiner data dictionaries exist, and you specify a non-NULL value for the first_scn parameter during capture process creation, then the new capture process creates a new LogMiner data dictionary the first time it is started. In this case, before you create the new capture process, you must run the BUILD procedure in the DBMS_CAPTURE_ADM package on the source database. The BUILD procedure generates a corresponding valid first scn value that you can specify when you create the new capture process. You can find a first SCN generated by the BUILD procedure by running the following query:

```
COLUMN FIRST_CHANGE# HEADING 'First SCN' FORMAT 999999999
COLUMN NAME HEADING 'Log File Name' FORMAT A50

SELECT DISTINCT FIRST_CHANGE#, NAME FROM V$ARCHIVED_LOG
  WHERE DICTIONARY_BEGIN = 'YES';
```

This query can return more than one row if the BUILD procedure was run more than once.

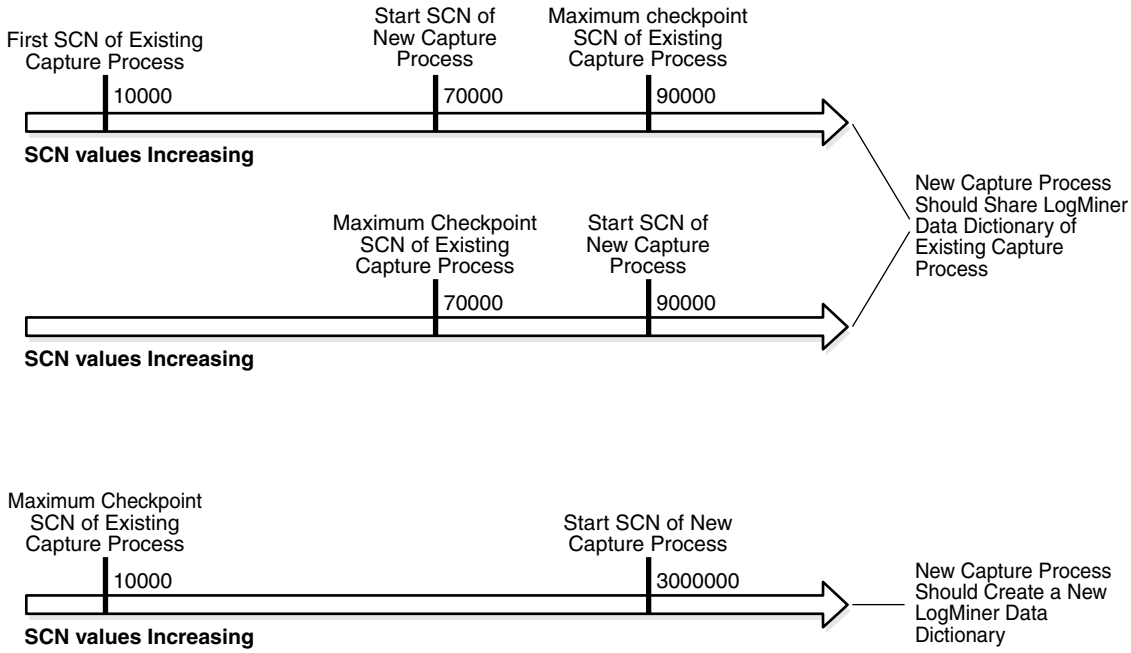
The most important factor to consider when deciding whether a new capture process should share an existing LogMiner data dictionary or create a new one is the difference between the maximum checkpoint SCN values of the existing capture processes and the **start SCN** of the new capture process. If the new capture process shares a LogMiner data dictionary, then it must scan the redo log from the point of the maximum checkpoint SCN of the shared LogMiner data dictionary onward, even though the new capture process cannot capture changes prior to its first SCN. If the start SCN of the new capture process is much higher than the maximum checkpoint SCN of the existing capture process, then the new capture process must scan a large amount of redo data before it reaches its start SCN.

A capture process creates a new LogMiner data dictionary when the `first_scn` parameter is non-`NULL` during capture process creation. Follow these guidelines when you decide whether a new capture process should share an existing LogMiner data dictionary or create a new one:

- If one or more maximum checkpoint SCN values is greater than the start SCN you want to specify, and if this start SCN is greater than the first SCN of one or more existing capture processes, then it might be better to share the LogMiner data dictionary of an existing capture process. In this case, you can assume there is a checkpoint SCN that is less than the start SCN and that the difference between this checkpoint SCN and the start SCN is small. The new capture process will begin scanning the redo log from this checkpoint SCN and will catch up to the start SCN quickly.
- If no maximum checkpoint SCN is greater than the start SCN, and if the difference between the maximum checkpoint SCN and the start SCN is small, then it might be better to share the LogMiner data dictionary of an existing capture process. The new capture process will begin scanning the redo log from the maximum checkpoint SCN, but it will catch up to the start SCN quickly.
- If no maximum checkpoint SCN is greater than the start SCN, and if the difference between the highest maximum checkpoint SCN and the start SCN is large, then it might take a long time for the capture process to catch up to the start SCN. In this case, it might be better for the new capture process to create a new LogMiner data dictionary. It will take some time to create the new LogMiner data dictionary when the new capture process is first started, but the capture process can specify the same value for its first SCN and start SCN, and thereby avoid scanning a large amount of redo data unnecessarily.

Figure 2–5 illustrates these guidelines.

Figure 2–5 Deciding Whether to Share a LogMiner Data Dictionary



Note:

- If you create a capture process using one of the procedures in the DBMS_STREAMS_ADM package, then it is the same as specifying NULL for the first_scn and start_scn parameters in the CREATE_CAPTURE procedure.
- You must prepare database objects for **instantiation** if a new capture process will capture changes made to these database objects. This requirement holds even if the new capture process shares a LogMiner data dictionary with one or more other capture processes for which these database objects have been prepared for instantiation.

See Also:

- "First SCN and Start SCN" on page 2-19
- "Capture Process Checkpoints" on page 2-25

First SCN and Start SCN Specifications During Capture Process Creation

When you create a capture process using the CREATE_CAPTURE procedure in the DBMS_CAPTURE_ADM package, you can specify the first SCN and start SCN for the capture process. The first SCN is the lowest SCN in the redo log from which a capture process can capture changes, and it should be obtained through a data dictionary build or a query on the V$ARCHIVED_LOG dynamic performance view. The start SCN is the SCN from which a capture process begins to capture changes. The start SCN must be equal to or greater than the first SCN.

A capture process scans the redo data from the first SCN or an existing capture process checkpoint forward, even if the start SCN is higher than the first SCN or the checkpoint SCN. In this case, the capture process does not capture any changes in the redo data before the start SCN. Oracle recommends that, at capture process creation time, the difference between the first SCN and start SCN be as small as possible to keep the amount of redo scanned by the capture process to a minimum.

In some cases, the behavior of the capture process is different depending on the settings of these SCN values and on whether the capture process is local or downstream.

Note: When you create a capture process using the DBMS_STREAMS_ADM package, both the first SCN and the start SCN are set to NULL during capture process creation.

The following sections describe capture process behavior for SCN value settings:

- Non-NULL First SCN and NULL Start SCN for a Local or Downstream Capture Process
- Non-NULL First SCN and Non-NULL Start SCN for a Local or Downstream Capture Process
- NULL First SCN and Non-NULL Start SCN for a Local Capture Process
- NULL First SCN and Non-NULL Start SCN for a Downstream Capture Process
- NULL First SCN and NULL Start SCN

Non-NULL First SCN and NULL Start SCN for a Local or Downstream Capture Process The new capture process is created at the local database with a new LogMiner session starting from the value specified for the `first_scn` parameter. The start SCN is set to the specified first SCN value automatically, and the new capture process does not capture changes that were made before this SCN.

The `BUILD` procedure in the `DBMS_CAPTURE_ADM` package is not run automatically. This procedure must have been run at least once before on the source database, and the specified first SCN must correspond to the SCN value of a previous build that is still available in the redo log. When the new capture process is started for the first time, it creates a new LogMiner data dictionary using the data dictionary information in the redo log. If the `BUILD` procedure in the `DBMS_CAPTURE_ADM` package has not been run at least once on the source database, then an error is raised when the capture process is started.

Capture process behavior is the same for a **local capture process** and a **downstream capture process** created with these SCN settings, except that a local capture process is created at the source database and a downstream capture process is created at the downstream database.

Non-NULL First SCN and Non-NULL Start SCN for a Local or Downstream Capture Process If the specified value for the `start_scn` parameter is greater than or equal to the specified value for the `first_scn` parameter, then the new capture process is created at the local database with a new LogMiner session starting from the specified first SCN. In this case, the new capture process does not capture changes that were made before the specified start SCN. If the specified value for the `start_scn` parameter is less than the specified value for the `first_scn` parameter, then an error is raised.

The `BUILD` procedure in the `DBMS_CAPTURE_ADM` package is not run automatically. This procedure must have been called at least once before on the source database, and the specified `first_scn` must correspond to the SCN value of a previous build that is still available in the redo log. When the new capture process is started for the first time, it creates a new LogMiner data dictionary using the data dictionary information in the redo log. If the `BUILD` procedure in the `DBMS_CAPTURE_ADM` package has not been run at least once on the source database, then an error is raised.

Capture process behavior is the same for a local capture process and a downstream capture process created with these SCN settings, except that a local capture process is created at the source database and a downstream capture process is created at the downstream database.

NULL First SCN and Non-NULL Start SCN for a Local Capture Process The new capture process creates a new LogMiner data dictionary if either one of the following conditions is true:

- There is no existing capture process for the local source database, and the specified value for the `start_scn` parameter is greater than or equal to the current SCN for the database.
- There are existing capture processes, but none of the capture processes have taken a checkpoint yet, and the specified value for the `start_scn` parameter is greater than or equal to the current SCN for the database.

In either of these cases, the `BUILD` procedure in the `DBMS_CAPTURE_ADM` package is run during capture process creation. The new capture process uses the resulting build of the source data dictionary in the redo log to create a LogMiner data dictionary the first time it is started, and the first SCN corresponds to the SCN of the data dictionary build.

However, if there is at least one existing local capture process for the local source database that has taken a checkpoint, then the new capture process shares an existing LogMiner data dictionary with one or more of the existing capture processes. In this case, a capture process with a first SCN that is lower than or equal to the specified start SCN must have been started successfully at least once.

If there is no existing capture process for the local source database (or if no existing capture processes have taken a checkpoint yet), and the specified start SCN is less than the current SCN for the database, then an error is raised.

NULL First SCN and Non-NULL Start SCN for a Downstream Capture Process If the `use_database_link` parameter is set to `true` during capture process creation, then the database link is used to obtain the current SCN of the source database. In this case, the new capture process creates a new LogMiner data dictionary if either one of the following conditions is true:

- There is no existing capture process that captures changes to the source database at the downstream database, and the specified value for the `start_scn` parameter is greater than or equal to the current SCN for the source database.
- There are existing capture processes that capture changes to the source database at the downstream database, but none of the capture processes have taken a checkpoint yet, and the specified value for the `start_scn` parameter is greater than or equal to the current SCN for the source database.

In either of these cases, the `BUILD` procedure in the `DBMS_CAPTURE_ADM` package is run during capture process creation. The first time you start the new capture process, it uses the resulting build of the source data dictionary in the redo log files copied to the downstream database to create a LogMiner data dictionary. Here, the first SCN for the new capture process corresponds to the SCN of the data dictionary build.

However, if at least one existing capture process has taken a checkpoint and captures changes to the source database at the downstream database, then the new capture process shares an existing LogMiner data dictionary with one or more of these existing capture processes, regardless of the `use_database_link` parameter setting. In this case, one of these existing capture processes with a first SCN that is lower than or equal to the specified start SCN must have been started successfully at least once.

If the `use_database_link` parameter is set to `true` during capture process creation, there is no existing capture process that captures changes to the source database at the downstream database (or no existing capture process has taken a checkpoint), and the specified `start_scn` parameter value is less than the current SCN for the source database, then an error is raised.

If the `use_database_link` parameter is set to `false` during capture process creation and there is no existing capture process that captures changes to the source database at the downstream database (or no existing capture process has taken a checkpoint), then an error is raised.

NULL First SCN and NULL Start SCN The behavior is the same as setting the `first_scn` parameter to `NULL` and setting the `start_scn` parameter to the current SCN of the source database.

See Also:

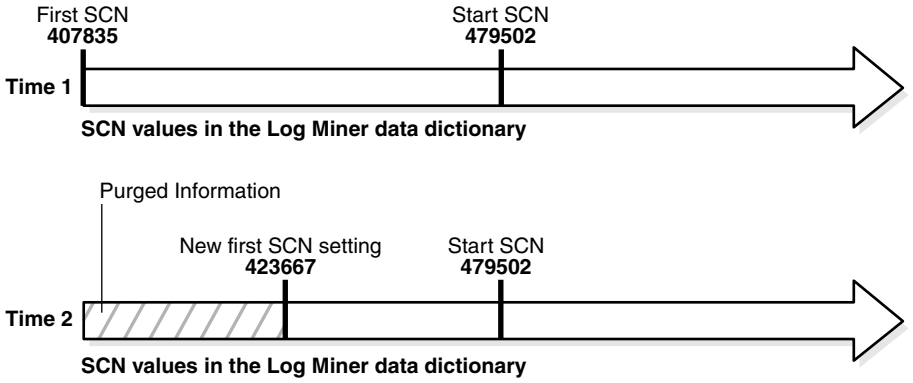
- "NULL First SCN and Non-NULL Start SCN for a Local Capture Process" on page 2-34
- "NULL First SCN and Non-NULL Start SCN for a Downstream Capture Process" on page 2-35

A New First SCN Value and Purged LogMiner Data Dictionary Information

When you reset the **first SCN** value for an existing capture process, Oracle automatically purges **LogMiner data dictionary** information prior to the new first SCN setting. If the **start SCN** for a capture process corresponds to information that has been purged, then Oracle automatically resets the start SCN to the same value as the first SCN. However, if the start SCN is higher than the new first SCN setting, then the start SCN remains unchanged.

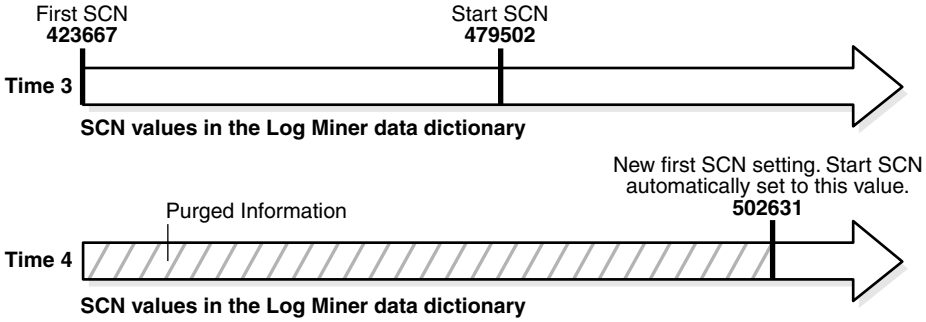
Figure 2–6 shows how Oracle automatically purges LogMiner data dictionary information prior to a new first SCN setting, and how the start SCN is not changed if it is higher than the new first SCN setting.

Figure 2–6 Start SCN Higher than Reset First SCN



Given this example, if the first SCN is reset again to a value higher than the start SCN value for a capture process, then the start SCN no longer corresponds to existing information in the LogMiner data dictionary. Figure 2–7 shows how Oracle resets the start SCN automatically if it is lower than a new first SCN setting.

Figure 2–7 Start SCN Lower than Reset First SCN



As you can see, the first SCN and start SCN for a capture process can continually increase over time, and, as the first SCN moves forward, it might no longer correspond to an SCN established by the `DBMS_CAPTURE_ADM.BUILD` procedure.

See Also:

- "First SCN and Start SCN" on page 2-19
- "Setting the Start SCN for an Existing Capture Process" on page 11-31
- The `DBMS_CAPTURE_ADM.ALTER_CAPTURE` procedure in the *Oracle Database PL/SQL Packages and Types Reference* for information about altering a capture process

The Streams Data Dictionary

Propagations and **apply process**es use a **Streams data dictionary** to keep track of the database objects from a particular **source database**. A Streams data dictionary is populated whenever one or more database objects are prepared for **instantiation** at a source database. Specifically, when a database object is prepared for instantiation, it is recorded in the redo log. When a capture process scans the redo log, it uses this information to populate the local Streams data dictionary for the source database. In the case of local capture, this Streams data dictionary is at the source database. In the case of downstream capture, this Streams data dictionary is at the **downstream database**.

When you prepare a database object for instantiation, you are informing Streams that information about the database object is needed by **propagation**s that propagate changes to the database object and apply processes that apply changes to the database object. Any database that propagates or applies these changes requires a Streams data dictionary for the source database where the changes originated.

After an object has been prepared for instantiation, the local Streams data dictionary is updated when a DDL statement on the object is processed by a capture process. In addition, an internal **message** containing information about this DDL statement is captured and placed in the **queue** for the capture process. Propagations can then propagate these internal messages to **destination queue**s at databases.

A Streams data dictionary is multiversioned. If a database has multiple propagations and apply processes, then all of them use the same Streams data dictionary for a particular source database. A database can contain only one Streams data dictionary for a particular source database, but it can contain multiple Streams data dictionaries if it propagates or applies changes from multiple source databases.

See Also:

- *Oracle Streams Replication Administrator's Guide* for more information about instantiation
- "Streams Data Dictionary for Propagations" on page 3-26
- "Streams Data Dictionary for an Apply Process" on page 4-13

ARCHIVELOG Mode and a Capture Process

The following list describes how different types of capture processes read the redo data:

- A **local capture process** reads online redo logs whenever possible and archived redo log files otherwise. Therefore, the **source database** must be running in ARCHIVELOG mode when a local capture process is configured to capture changes.
- A **real-time downstream capture process** reads online redo data from its source database whenever possible and archived redo log files that contain redo data from the source database otherwise. In this case, the redo data from the source database is stored in the standby redo log at the **downstream database**, and the archiver at the downstream database archives the redo data in the standby redo log. Therefore, both the source database and the downstream database must be running in ARCHIVELOG mode when a real-time downstream capture process is configured to capture changes.
- An **archived-log downstream capture process** always reads archived redo log files from its source database. Therefore, the source database must be running in ARCHIVELOG mode when an archived-log downstream capture process is configured to capture changes.

You can query the REQUIRED_CHECKPOINT_SCN column in the DBA_CAPTURE data dictionary view to determine the **required checkpoint SCN** for a capture process. When the capture process is restarted, it scans the redo log from the required checkpoint SCN forward. Therefore, the redo log file that includes the required checkpoint SCN, and all subsequent redo log files, must be available to the capture process.

You must keep an archived redo log file available until you are certain that no capture process will need that file. The **first SCN** for a capture process can be reset to a higher value, but it cannot be reset to a lower value. Therefore, a capture process will never need the redo log files that contain information prior to its first SCN. Query the DBA_LOGMNR_PURGED_LOG data dictionary view to determine which archived redo log files will never be needed by any capture process.

When a local capture process falls behind, there is a seamless transition from reading an online redo log to reading an archived redo log, and, when a local capture process catches up, there is a seamless transition from reading an archived redo log to reading an online redo log. Similarly, when a real-time downstream capture process falls behind, there is a seamless transition from reading the standby redo log to reading an archived redo log, and, when a real-time downstream capture process catches up, there is a seamless transition from reading an archived redo log to reading the standby redo log.

Note: At a downstream database in a downstream capture configuration, log files from a remote source database should be kept separate from local database log files. In addition, if the downstream database contains log files from multiple source databases, then the log files from each source database should be kept separate from each other.

See Also:

- *Oracle Database Administrator's Guide* for information about running a database in `ARCHIVELOG` mode
- "Displaying SCN Values for Each Redo Log File Used by Each Capture Process" on page 20-9 for a query that determines which redo log files are no longer needed

RMAN and Archived Redo Log Files Required by a Capture Process

Some Recovery Manager (RMAN) commands delete archived redo log files. If one of these RMAN commands is used on a database that is running one or more local capture processes, then the RMAN command does not delete archived redo log files that are needed by a local capture process. That is, the RMAN command does not delete archived redo log files that contain changes with SCN values that are equal to or greater than the required checkpoint SCN for a local capture process.

The following RMAN commands delete archived redo log files:

- The RMAN command `DELETE OBSOLETE` permanently purges the archived redo log files that are no longer needed. This command only deletes the archived redo log files in which all of the changes are less than the required checkpoint SCN for a local capture process.
- The RMAN command `BACKUP ARCHIVELOG ALL DELETE INPUT` copies the archived redo log files and deletes the original files after completing the backup. This command only deletes the archived redo log files in which all of the changes are less than the required checkpoint SCN for a local capture process. If archived redo log files are not deleted because they contain changes required by a capture process, then RMAN display a warning message about skipping the delete operation for these files.

If a database is a source database for a **downstream capture process**, then these RMAN commands might delete archived redo log files that have not been transferred to the downstream database and are required by a downstream capture process. Therefore, before running these commands on the source database, make sure any archived redo log files needed by a downstream database have been transferred to the downstream database.

Note: The flash recovery area feature of RMAN might delete archived redo log files that are required by a capture process.

See Also:

- "Are Required Redo Log Files Missing?" on page 18-3 for information about determining whether a capture process is missing required archived redo log files and for information correcting this problem. This section also contains information about flash recovery area and local capture processes.
- *Oracle Database Backup and Recovery Advanced User's Guide* and *Oracle Database Backup and Recovery Reference* for more information about RMAN

Capture Process Parameters

After creation, a capture process is disabled so that you can set the capture process parameters for your environment before starting it for the first time. Capture process parameters control the way a capture process operates. For example, the `time_limit` capture process parameter specifies the amount of time a capture process runs before it is shut down automatically.

See Also:

- "Setting a Capture Process Parameter" on page 11-28
- This section does not discuss all of the available capture process parameters. See the `DBMS_CAPTURE_ADM.SET_PARAMETER` procedure in the *Oracle Database PL/SQL Packages and Types Reference* for detailed information about all of the capture process parameters.

Capture Process Parallelism

The `parallelism` capture process parameter controls the number of **preparer server**s used by a capture process. The preparer servers concurrently format changes found in the redo log into LCRs. Each **reader server**, preparer server, and **builder server** is a parallel execution server, and the number of preparer servers equals the number specified for the `parallelism` capture process parameter. So, if `parallelism` is set to `5`, then a capture process uses a total of seven parallel execution servers, assuming seven parallel execution servers are available: one reader server, five preparer servers, and one builder server.

Note:

- Resetting the `parallelism` parameter automatically stops and restarts the capture process.
- Setting the `parallelism` parameter to a number higher than the number of available parallel execution servers might disable the capture process. Make sure the `PROCESSES` and `PARALLEL_MAX_SERVERS` initialization parameters are set appropriately when you set the `parallelism` capture process parameter.

See Also: "Capture Process Components" on page 2-23 for more information about preparer servers

Automatic Restart of a Capture Process

You can configure a capture process to stop automatically when it reaches certain limits. The `time_limit` capture process parameter specifies the amount of time a capture process runs, and the `message_limit` capture process parameter specifies the number of **message**s a capture process can capture. The capture process stops automatically when it reaches one of these limits.

The `disable_on_limit` parameter controls whether a capture process becomes disabled or restarts when it reaches a limit. If you set the `disable_on_limit` parameter to `y`, then the capture process is disabled when it reaches a limit and does not restart until you restart it explicitly. If, however, you set the `disable_on_limit` parameter to `n`, then the capture process stops and restarts automatically when it reaches a limit.

When a capture process is restarted, it starts to capture changes at the point where it last stopped. A restarted capture process gets a new session identifier, and the parallel execution servers associated with the capture process also get new session identifiers. However, the capture process number (`cnnn`) remains the same.

Capture Process Rule Evaluation

A capture process evaluates changes it finds in the redo log against its positive and **negative rule set**s. The capture process evaluates a change against the negative rule set first. If one or more **rule**s in the negative rule set evaluate to `TRUE` for the change, then the change is discarded, but if no rule in the negative rule set evaluates to `TRUE` for the change, then the change satisfies the negative rule set. When a change satisfies the negative rule set for a capture process, the capture process evaluates the change against its **positive rule set**. If one or more rules in the positive rule set evaluate to `TRUE` for the change, then the change satisfies the positive rule set, but if no rule in the positive rule set evaluates to `TRUE` for the change, then the change is discarded. If a capture process only has one rule set, then it evaluates changes against this one rule set only.

A running capture process completes the following series of actions to capture changes:

1. Finds changes in the redo log.
2. Performs prefiltering of the changes in the redo log. During this step, a capture process evaluates rules in its rule sets at a basic level to place changes found in the redo log into two categories: changes that should be converted into LCRs and changes that should not be converted into LCRs. Prefiltering is done in two phases. In the first phase, information that can be evaluated during prefiltering includes schema name, object name, and command type. If more information is needed to determine whether a change should be converted into an LCR, then information that can be evaluated during the second phase of prefiltering includes **tag** values and column values when appropriate.

   Prefiltering is a safe optimization done with incomplete information. This step identifies relevant changes to be processed subsequently, such that:

   - A capture process converts a change into an LCR if the change satisfies the capture process rule sets. In this case, proceed to Step 3.
   - A capture process does not convert a change into an LCR if the change does not satisfy the capture process rule sets.
   - Regarding `MAYBE` evaluations, the rule evaluation proceeds as follows:
     - If a change evaluates to `MAYBE` against both the positive and negative rule set for a capture process, then the capture process might not have enough information to determine whether the change will definitely satisfy both of its rule sets. In this case, further evaluation is necessary. Proceed to Step 3.
     - If the change evaluates to `FALSE` against the negative rule set and `MAYBE` against the positive rule set for the capture process, then the capture process might not have enough information to determine whether the change will definitely satisfy both of its rule sets. In this case, further evaluation is necessary. Proceed to Step 3.

- If the change evaluates to MAYBE against the negative rule set and TRUE against the positive rule set for the capture process, then the capture process might not have enough information to determine whether the change will definitely satisfy both of its rule sets. In this case, further evaluation is necessary. Proceed to Step 3.
- If the change evaluates to TRUE against the negative rule set and MAYBE against the positive rule set for the capture process, then the capture process discards the change.
- If the change evaluates to MAYBE against the negative rule set and FALSE against the positive rule set for the capture process, then the capture process discards the change.

3. Converts changes that satisfy, or might satisfy, the capture process rule sets into LCRs based on prefiltering.
4. Performs LCR filtering. During this step, a capture process evaluates rules regarding information in each LCR to separate the LCRs into two categories: LCRs that should be enqueued and LCRs that should be discarded.
5. Discards the LCRs that should not be enqueued because they did not satisfy the capture process rule sets.
6. Enqueues the remaining **captured message**s into the **queue** associated with the capture process.

For example, suppose the following rule is defined in the positive rule set for a capture process: Capture changes to the hr.employees table where the department_id is 50. No other rules are defined for the capture process, and the parallelism parameter for the capture process is set to 1.

Given this rule, suppose an UPDATE statement on the hr.employees table changes 50 rows in the table. The capture process performs the following series of actions for each row change:

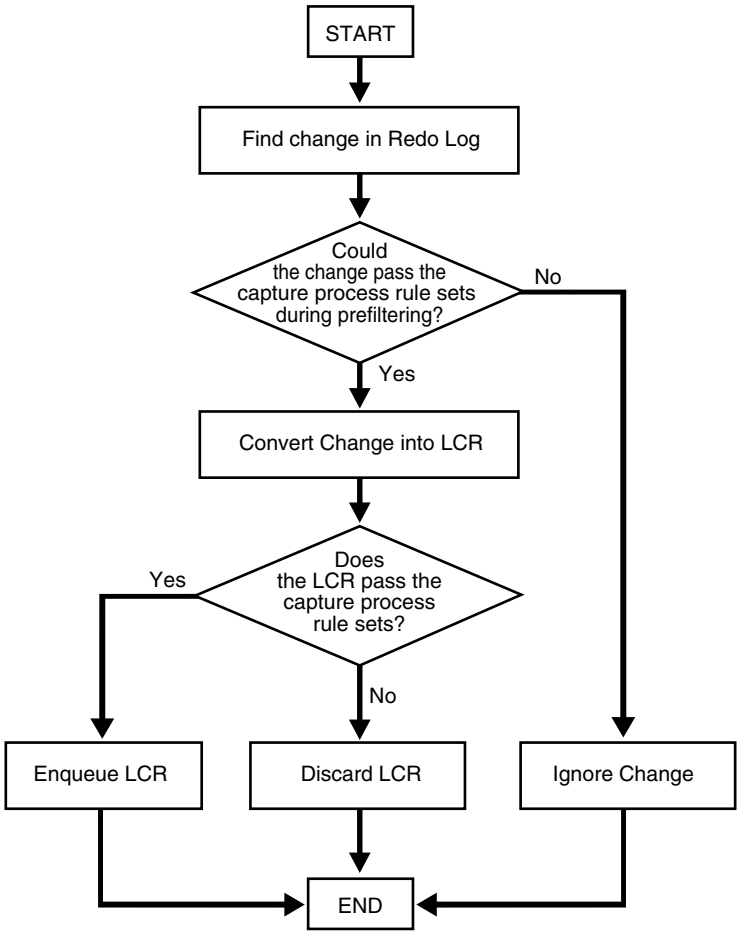
1. Finds the next change resulting from the UPDATE statement in the redo log.
2. Determines that the change resulted from an UPDATE statement to the hr.employees table and must be captured. If the change was made to a different table, then the capture process ignores the change.
3. Captures the change and converts it into an LCR.
4. Filters the LCR to determine whether it involves a row where the department_id is 50.
5. Either enqueues the LCR into the queue associated with the capture process if it involves a row where the department_id is 50, or discards the LCR if it involves a row where the department_id is not 50 or is missing.

See Also:

- "Capture Process Components" on page 2-23
- Chapter 6, "How Rules Are Used in Streams" for more information about rule sets for **Streams client**s and for information about how messages satisfy rule sets

Figure 2–8 illustrates capture process rule evaluation in a flowchart.

Figure 2–8 Flowchart Showing Capture Process Rule Evaluation



Persistent Capture Process Status Upon Database Restart

A capture process maintains a persistent status when the database running the capture process is shut down and restarted. For example, if a capture process is enabled when the database is shut down, then the capture process automatically starts when the database is restarted. Similarly, if a capture process is disabled or aborted when a database is shut down, then the capture process is not started and retains the disabled or aborted status when the database is restarted.

3

Streams Staging and Propagation

This chapter explains the concepts relating to staging **message**s in a **queue** and propagating messages from one queue to another.

This chapter contains these topics:

- Introduction to Message Staging and Propagation
- Captured and User-Enqueued Messages in an ANYDATA Queue
- Message Propagation Between Queues
- Messaging Clients
- ANYDATA Queues and User Messages
- Buffered Messaging and Streams Clients
- Queues and Oracle Real Application Clusters
- Commit-Time Queues
- Streams Staging and Propagation Architecture

See Also: Chapter 12, "Managing Staging and Propagation"

Introduction to Message Staging and Propagation

Streams uses **queue**s to stage **message**s. A queue of `ANYDATA` type can stage messages of almost any type and is called a **ANYDATA queue**. A **typed queue** can store messages of a specific type. **Streams client**s always use `ANYDATA` queues.

In Streams, two types of messages can be encapsulated into an `ANYDATA` object and staged in an `ANYDATA` queue: logical change records (LCRs) and **user message**s. An LCR is an object that contains information about a change to a database object. A user message is a message of a user-defined type created by users or applications. Both types of messages can be used for information sharing within a single database or between databases.

In a messaging environment, both `ANYDATA` queues and **typed queue**s can be used to stage messages of a specific type. Publishing applications can enqueue messages into a single queue, and subscribing applications can dequeue these messages.

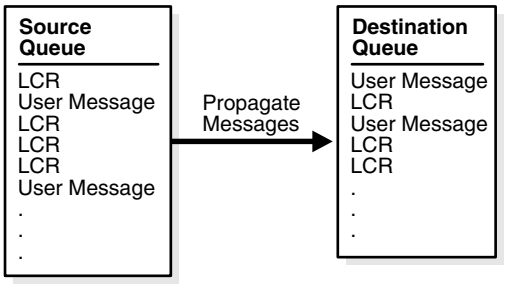
Staged messages can be consumed or propagated, or both. Staged messages can be consumed by an **apply process**, by a **messaging client**, or by a user application. A running apply process implicitly dequeues messages, but messaging clients and user applications explicitly dequeue messages. Even after a message is consumed, it can remain in the queue if you also have configured a Streams **propagation** to propagate, or send, the message to one or more other queues or if message retention is specified

for **user-enqueued message**s. Message retention does not apply to LCRs captured by a **capture process**.

The queues to which messages are propagated can reside in the same database or in different databases than the queue from which the messages are propagated. In either case, the queue from which the messages are propagated is called the **source queue**, and the queue that receives the messages is called the **destination queue**. There can be a one-to-many, many-to-one, or many-to-many relationship between source and destination queues.

Figure 3–1 shows propagation from a source queue to a destination queue.

Figure 3–1 Propagation from a Source Queue to a Destination Queue



You can create, alter, and drop a propagation, and you can define propagation **rule**s that control which messages are propagated. The user who owns the source queue is the user who propagates messages, and this user must have the necessary privileges to propagate messages. These privileges include the following:

- `EXECUTE` privilege on the **rule set**s used by the propagation
- `EXECUTE` privilege on all **custom rule-based transformation** functions used in the rule sets
- Enqueue privilege on the destination queue if the destination queue is in the same database

If the propagation propagates messages to a destination queue in a remote database, then the owner of the source queue must be able to use the database link used by the propagation, and the user to which the database link connects at the remote database must have enqueue privilege on the destination queue.

Note: Connection qualifiers cannot be specified in the database links that are used by Streams propagations.

See Also:

- "Logical Change Records (LCRs)" on page 2-2
- *Oracle Streams Advanced Queuing User's Guide and Reference* for more information about message retention for user-enqueued messages

Captured and User-Enqueued Messages in an ANYDATA Queue

Messages can be enqueued into an ANYDATA queue in two ways:

- A **capture process** enqueues captured changes in the form of **message**s containing LCRs. A message containing an LCR that was originally captured and enqueued by a capture process is called a **captured message**.
- A user application enqueues **user message**s encapsulated in objects of type ANYDATA. These user messages can contain LCRs or any other type of information. Any user message that was explicitly enqueued by a user or an application is called a **user-enqueued message**. Messages that were enqueued by a user procedure called from an **apply process** are also user-enqueued messages.

So, each captured message contains an LCR, but a user-enqueued message might or might not contain an LCR. Propagating a captured message or a user-enqueued message enqueues the message into the **destination queue**.

Messages can be dequeued from an ANYDATA queue in two ways:

- An apply process dequeues either captured or user-enqueued messages. If the message contains an LCR, then the apply process can either apply it directly or call a user-specified procedure for processing. If the message does not contain an LCR, then the apply process can invoke a user-specified procedure called a **message handler** to process it. In addition, captured messages that are dequeued by an apply process and then enqueued using the SET_ENQUEUE_DESTINATION procedure in the DBMS_APPLY_ADM package are user-enqueued messages.
- A user application explicitly dequeues user-enqueued messages and processes them. The user application might or might not use a Streams messaging client. Captured messages cannot be dequeued by a user application. Captured messages must be dequeued by an apply process. However, if a user procedure called by an apply process explicitly enqueues a message, then the message is a user-enqueued message and can be explicitly dequeued, even if the message was originally a captured message.

The dequeued messages might have originated at the same database where they are dequeued, or they might have originated at a different database.

See Also:

- Chapter 2, "Streams Capture Process" for more information about the capture process
- "Messaging Clients" on page 3-10
- Chapter 4, "Streams Apply Process" for more information about the apply process
- *Oracle Streams Advanced Queuing User's Guide and Reference* for information about enqueuing messages into a queue
- *Oracle Streams Replication Administrator's Guide* for more information about managing LCRs

Message Propagation Between Queues

You can use Streams to configure **message** propagation between two **queue**s, which can reside in different databases. Streams uses job queues to propagate messages.

A **propagation** is always between a **source queue** and a **destination queue**. Although propagation is always between two queues, a single queue can participate in many propagations. That is, a single source queue can propagate messages to multiple destination queues, and a single destination queue can receive messages from multiple source queues. However, only one propagation is allowed between a particular source queue and a particular destination queue. Also, a single queue can be a destination queue for some propagations and a source queue for other propagations.

A propagation can propagate all of the messages in a source queue to a destination queue, or a propagation can propagate only a subset of the messages. Also, a single propagation can propagate both **captured message**s and **user-enqueued message**s. You can use **rule**s to control which messages in the source queue are propagated to the destination queue and which messages are discarded.

Depending on how you set up your Streams environment, changes could be sent back to the site where they originated. You need to ensure that your environment is configured to avoid cycling a change in an endless loop. You can use Streams **tag**s to avoid such a **change cycling** loop.

Note: Propagations can propagate user-enqueued `ANYDATA` messages that encapsulate payloads of object types, varrays, or nested tables between databases only if the databases use the same character set.

See Also:

- "Managing Streams Propagations and Propagation Jobs" on page 12-6
- *Oracle Streams Advanced Queuing User's Guide and Reference* for detailed information about the propagation infrastructure in AQ
- *Oracle Streams Replication Administrator's Guide* for more information about Streams tags

Propagation Rules

A **propagation** either propagates or discards **message**s based on **rule**s that you define. For LCRs, each rule specifies the database objects and types of changes for which the rule evaluates to `TRUE`. You can place these rules in a **positive rule set** or a **negative rule set** used by the propagation.

If a rule evaluates to `TRUE` for a message, and the rule is in the positive rule set for a propagation, then the propagation propagates the change. If a rule evaluates to `TRUE` for a message, and the rule is in the negative rule set for a propagation, then the propagation discards the change. If a propagation has both a positive and a negative rule set, then the negative rule set is always evaluated first.

You can specify propagation rules for LCRs at the following levels:

- A table rule propagates or discards either row changes resulting from DML changes or DDL changes to a particular table. Subset rules are table rules that include a subset of the row changes to a particular table.
- A schema rule propagates or discards either row changes resulting from DML changes or DDL changes to the database objects in a particular schema.
- A global rule propagates or discards either all row changes resulting from DML changes or all DDL changes in the source **queue**.

For non-LCR messages, you can create your own rules to control propagation.

A queue subscriber that specifies a condition causes the system to generate a rule. The rule sets for all subscribers to a queue are combined into a single system-generated rule set to make subscription more efficient.

See Also:

- Chapter 5, "Rules"
- Chapter 6, "How Rules Are Used in Streams"

Queue-to-Queue Propagations

A **propagation** can be queue-to-queue or queue-to-database link (queue-to-dblink). A queue-to-queue propagation always has its own exclusive **propagation job** to propagate **message**s from the **source queue** to the **destination queue**. Because each propagation job has its own **propagation schedule**, the propagation schedule of each queue-to-queue propagation can be managed separately. Even when multiple queue-to-queue propagations use the same database link, you can enable, disable, or set the propagation schedule for each queue-to-queue propagation separately. Propagation jobs are described in detail later in this chapter.

A single database link can be used by multiple queue-to-queue propagations. The database link must be created with the service name specified as the global name of the database that contains the destination queue.

In contrast, a queue-to-dblink propagation shares a propagation job with other queue-to-dblink propagations from the same source queue that use the same database link. Therefore, these propagations share the same propagation schedule, and any change to the propagation schedule affects all of the queue-to-dblink propagations from the same source queue that use the database link.

Queue-to-queue propagation connects to the destination queue service when one exists. Currently, a queue service is created when the database is a Real Application Clusters (RAC) database and the queue is a **buffered queue**. Because the queue service always runs on the owner instance of the queue, transparent failover can occur when RAC instances fail. When multiple queue-to-queue propagations use a single database link, the connect description for each queue-to-queue propagation changes automatically to propagate messages to the correct destination queue. In contrast, queue-to-dblink propagations require you to repoint your database links if the owner instance in a RAC database that contains the destination queue for the propagation fails.

Note: To use queue-to-queue propagation, the compatibility level must be 10.2.0 or higher for each database that contains a queue involved in the propagation.

See Also:

- "Queues and Oracle Real Application Clusters" on page 3-13
- "Propagation Jobs" on page 3-22
- Chapter 12, "Managing Staging and Propagation" for information about creating queue-to-queue propagations and managing the propagation job for a queue-to-queue propagation

Ensured Message Delivery

A **user-enqueued message** is propagated successfully to a **destination queue** when the enqueue into the destination queue is committed. A **captured message** is propagated successfully to a destination queue when both of the following actions are completed:

- The **message** is processed by all relevant **apply process**es associated with the destination queue.
- The message is propagated successfully from the destination queue to all of its relevant destination queues.

When a message is successfully propagated between two `ANYDATA` queues, the destination queue acknowledges successful propagation of the message. If the **source queue** is configured to propagate a message to multiple destination queues, then the message remains in the source queue until each destination queue has sent confirmation of message propagation to the source queue. When each destination queue acknowledges successful propagation of the message, and all local consumers in the source queue database have consumed the message, the source queue can drop the message.

This confirmation system ensures that messages are always propagated from the source queue to the destination queue, but, in some configurations, the source queue can grow larger than an optimal size. When a source queue grows, it uses more SGA memory and might use more disk space.

There are two common reasons for source-queue growth:

- If a message cannot be propagated to a specified destination queue for some reason (such as a network problem), then the message will remain in the source queue until the destination queue becomes available. This situation could cause the source queue to grow large. So, you should monitor your queues regularly to detect problems early.
- Suppose a source queue is propagating captured messages to multiple destination queues, and one or more **destination database**s acknowledge successful propagation of messages much more slowly than the other queues. In this case, the source queue can grow because the slower destination databases create a backlog of messages that have already been acknowledged by the faster destination databases. In such an environment, consider creating more than one **capture process** to capture changes at the **source database**. Doing so lets you use one source queue for the slower destination databases and another source queue for the faster destination databases.

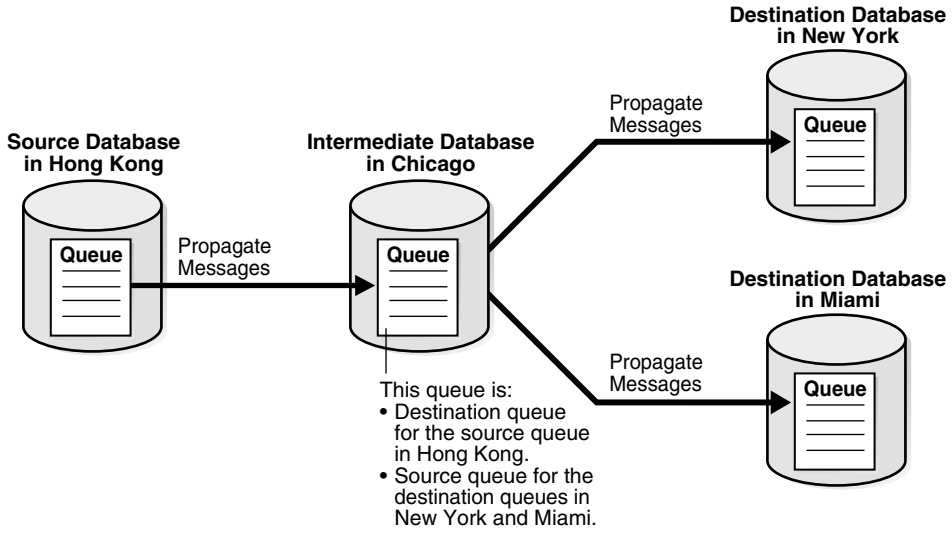
See Also:

- Chapter 2, "Streams Capture Process"
- "Monitoring ANYDATA Queues and Messaging" on page 21-1

Directed Networks

A **directed network** is one in which propagated **messages** pass through one or more intermediate databases before arriving at a **destination database**. A message might or might not be processed by an **apply process** at an intermediate database. Using Streams, you can choose which messages are propagated to each destination database, and you can specify the route that messages will traverse on their way to a destination database. Figure 3–2 shows an example of a directed networks environment.

Figure 3–2 Example Directed Networks Environment



The advantage of using a directed network is that a **source database** does not need to have a physical network connection with a destination database. So, if you want messages to propagate from one database to another, but there is no direct network connection between the computers running these databases, then you can still propagate the messages without reconfiguring your network, as long as one or more intermediate databases connect the source database to the destination database.

If you use directed networks, and an intermediate site goes down for an extended period of time or is removed, then you might need to reconfigure the network and the Streams environment.

Queue Forwarding and Apply Forwarding

An intermediate database in a directed network can propagate messages using either queue forwarding or apply forwarding. **Queue forwarding** means that the messages being forwarded at an intermediate database are the messages received by the intermediate database. The source database for a message is the database where the message originated.

Apply forwarding means that the messages being forwarded at an intermediate database are first processed by an apply process. These messages are then recaptured by a **capture process** at the intermediate database and forwarded. When you use apply forwarding, the intermediate database becomes the new source database for the messages, because the messages are recaptured from the redo log generated there.

Consider the following differences between queue forwarding and apply forwarding when you plan your Streams environment:

- With queue forwarding, a message is propagated through the directed network without being changed, assuming there are no capture or propagation transformations. With apply forwarding, messages are applied and recaptured at intermediate databases and can be changed by **conflict resolution**, **apply handler**s, or apply transformations.
- With queue forwarding, a destination database must have a separate apply process to apply messages from each source database. With apply forwarding, fewer apply processes might be required at a destination database because recapturing of messages at intermediate databases can result in fewer source databases when changes reach a destination database.
- With queue forwarding, one or more intermediate databases are in place between a source database and a destination database. With apply forwarding, because messages are recaptured at intermediate databases, the source database for a message can be the same as the intermediate database connected directly with the destination database.

A single Streams environment can use a combination of queue forwarding and apply forwarding.

Advantages of Queue Forwarding Queue forwarding has the following advantages compared with apply forwarding:

- Performance might be improved because a message is captured only once.
- Less time might be required to propagate a message from the database where the message originated to the destination database, because the messages are not applied and recaptured at one or more intermediate databases. In other words, latency might be lower with queue forwarding.
- The origin of a message can be determined easily by running the `GET_SOURCE_DATABASE_NAME` member procedure on the LCR contained in the message. If you use apply forwarding, then determining the origin of a message requires the use of Streams **tag**s and apply handlers.
- Parallel apply might scale better and provide more throughput when separate apply processes are used because there are fewer dependencies, and because there are multiple apply coordinators and apply reader processes to perform the work.
- If one intermediate database goes down, then you can reroute the queues and reset the **start SCN** at the capture site to reconfigure end-to-end capture, propagation, and apply.

  If you use apply forwarding, then substantially more work might be required to reconfigure end-to-end capture, propagation, and apply of messages, because the destination database(s) downstream from the unavailable intermediate database were using the SCN information of this intermediate database. Without this SCN information, the destination databases cannot apply the changes properly.

Advantages of Apply Forwarding Apply forwarding has the following advantages compared with queue forwarding:

- A Streams environment might be easier to configure because each database can apply changes only from databases directly connected to it, rather than from multiple remote source databases.
- In a large Streams environment where intermediate databases apply changes, the environment might be easier to monitor and manage because fewer apply processes might be required. An intermediate database that applies changes must have one apply process for each source database from which it receives changes. In an apply forwarding environment, the source databases of an intermediate database are only the databases to which it is directly connected. In a queue forwarding environment, the source databases of an intermediate database are all of the other source databases in the environment, whether they are directly connected to the intermediate database or not.

See Also:

- Chapter 4, "Streams Apply Process"
- *Oracle Streams Replication Administrator's Guide* for an example of an environment that uses queue forwarding and for an example of an environment that uses apply forwarding

Binary File Propagation

You can propagate a binary file between databases by using Streams. To do so, you put one or more `BFILE` attributes in a **message** payload and then propagate the message to a remote **queue**. Each `BFILE` referenced in the payload is transferred to the remote database after the message is propagated, but before the message propagation is committed. The directory object and filename of each propagated `BFILE` are preserved, but you can map the directory object to different directories on the source and **destination database**s. The message payload can be a `BFILE` wrapped in an `ANYDATA` payload, or the message payload can be one or more `BFILE` attributes of an object wrapped in an `ANYDATA` payload.

The following are not supported in a message payload:

- One or more `BFILE` attributes in a varray
- A user-defined type object with an `ANYDATA` attribute that contains one or more `BFILE` attributes

Propagating a `BFILE` in Streams has the same restrictions as the procedure `DBMS_FILE_TRANSFER.PUT_FILE`.

See Also: *Oracle Database Concepts*, *Oracle Database Administrator's Guide*, and *Oracle Database PL/SQL Packages and Types Reference* for more information about transferring files with the `DBMS_FILE_TRANSFER` package

Messaging Clients

A **messaging client** dequeues **user-enqueued message**s when it is invoked by an application or a user. You use **rule**s to specify which user-enqueued messages in the **queue** are dequeued by a messaging client. These user-enqueued messages can be user-enqueued LCRs or user-enqueued messages.

You can create a messaging client by specifying `dequeue` for the `streams_type` parameter when you run one of the following procedures in the `DBMS_STREAMS_ADM` package:

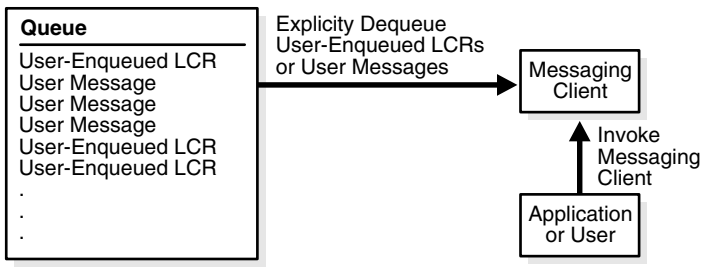
- `ADD_MESSAGE_RULE`
- `ADD_TABLE_RULES`
- `ADD_SUBSET_RULES`
- `ADD_SCHEMA_RULES`
- `ADD_GLOBAL_RULES`

When you create a messaging client, you specify the name of the messaging client and the `ANYDATA` queue from which the messaging client dequeues messages. These procedures can also add rules to the **positive rule set** or **negative rule set** of a messaging client. You specify the message type for each rule, and a single messaging client can dequeue messages of different types.

The user who creates a messaging client is granted the privileges to dequeue from the queue using the messaging client. This user is the **messaging client user**. The messaging client user can dequeue messages that satisfy the messaging client **rule set**s. A messaging client can be associated with only one user, but one user can be associated with many messaging clients.

Figure 3–3 shows a messaging client dequeuing user-enqueued messages.

Figure 3–3 Messaging Client



See Also:

- Chapter 6, "How Rules Are Used in Streams" for information about messaging clients and rules
- "Configuring a Messaging Client and Message Notification" on page 12-18

ANYDATA Queues and User Messages

Streams enables messaging with **queue**s of type ANYDATA. These queues can stage **user message**s whose payloads are of ANYDATA type. An ANYDATA payload can be a wrapper for payloads of different datatypes.

By using ANYDATA wrappers for **message** payloads, publishing applications can enqueue messages of different types into a single queue, and subscribing applications can dequeue these messages, either explicitly using a **messaging client** or an application, or implicitly using an **apply process**. If the subscribing application is remote, then the messages can be propagated to the remote site, and the subscribing application can dequeue the messages from a local queue in the remote database. Alternatively, a remote subscribing application can dequeue messages directly from the source queue using a variety of standard protocols, such as PL/SQL and OCI.

Streams includes the features of Advanced Queuing (AQ), which supports all the standard features of message queuing systems, including multiconsumer queues, publish and subscribe, content-based routing, internet propagation, transformations, and gateways to other messaging subsystems.

You can wrap almost any type of payload in an ANYDATA payload. To do this, you use the Convert*data_type* static functions of the ANYDATA type, where *data_type* is the type of object to wrap. These functions take the object as input and return an ANYDATA object.

You cannot enqueue ANYDATA payloads that contain payloads of the following types into an ANYDATA queue:

- CLOB
- NCLOB
- BLOB
- Object types with LOB attributes
- Object types that use type evolution or type inheritance

Note:

- Payloads of ROWID datatype cannot be wrapped in an ANYDATA wrapper. This restriction does not apply to payloads of UROWID datatype.
- A queue that can stage messages of only one particular type is called a **typed queue**.

See Also:

- "Managing a Streams Messaging Environment" on page 12-15
- "Wrapping User Message Payloads in an ANYDATA Wrapper and Enqueuing Them" on page 12-15
- *Oracle Streams Advanced Queuing User's Guide and Reference* for more information relating to `ANYDATA` queues, such as wrapping payloads in an `ANYDATA` wrapper, programmatic environments for enqueuing messages into and dequeuing messages from an `ANYDATA` queue, propagation, and user-defined types
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `ANYDATA` type

Buffered Messaging and Streams Clients

Buffered messaging enables users and applications to enqueue **message**s into and dequeue messages from a **buffered queue**. Propagations can propagate buffered messages from one buffered queue to another. Buffered messaging can improve the performance of a messaging environment by storing messages in memory instead of persistently on disk in a **queue table**. The following sections discuss how buffered messages interact with **Streams client**s:

- Buffered Messages and Capture Processes
- Buffered Messages and Propagations
- Buffered Messages and Apply Processes
- Buffered Messages and Messaging Clients

Note: To use buffered messaging, the compatibility level of the Oracle database must be 10.2.0 or higher.

See Also:

- "Buffered Queues" on page 3-21
- *Oracle Streams Advanced Queuing User's Guide and Reference* for detailed conceptual information about buffered messaging and for information about using buffered messaging

Buffered Messages and Capture Processes

Messages enqueued into a **buffered queue** by a **capture process** can be dequeued only by an **apply process**. Captured messages cannot be dequeued by users or applications.

Buffered Messages and Propagations

A **propagation** will propagate any **message**s in its **source queue** that satisfy its **rule set**s. These messages can be stored in a **buffered queue** or stored persistently in a **queue table**. A propagation can propagate both types of messages if the messages satisfy the rule sets used by the propagation.

Buffered Messages and Apply Processes

Apply processes can dequeue and process **message**s in a buffered queue. To dequeue messages in a **buffered queue** that were enqueued by a **capture process**, the **apply process** must be configured with the `apply_captured` parameter set to `true`. To dequeue messages in a buffered queue that were enqueued by a user or application, the apply process must be configured with the `apply_captured` parameter set to `false`. An apply process sends **user-enqueued message**s to its **message handler** for processing.

Buffered Messages and Messaging Clients

Currently, **messaging client**s cannot dequeue buffered **message**s. In addition, the `DBMS_STREAMS_MESSAGING` package cannot be used to enqueue messages into or dequeue messages from a **buffered queue**.

Queues and Oracle Real Application Clusters

You can configure a queue to stage **captured message**s and **user-enqueued message**s in an Oracle Real Application Clusters (RAC) environment, and **propagation**s can propagate these messages from one **queue** to another. In a RAC environment, only the owner instance can have a buffer for a queue, but different instances can have buffers for different queues. A **buffered queue** is System Global Area (SGA) memory associated with a queue. Buffered queues are discussed in more detail later in this chapter.

Streams processes and jobs support primary instance and secondary instance specifications for **queue table**s. If you use these specifications, then the secondary instance assumes ownership of a queue table when the primary instance becomes unavailable, and ownership is transferred back to the primary instance when it becomes available again. If both the primary and secondary instance for a queue table containing a **destination queue** become unavailable, then queue ownership is transferred automatically to another instance in the cluster. In this case, if the primary or secondary instance becomes available again, then ownership is transferred back to one of them accordingly. You can set primary and secondary instance specifications using the `ALTER_QUEUE_TABLE` procedure in the `DBMS_AQADM` package.

Each **capture process** and **apply process** is started on the owner instance for its queue, even if the start procedure is run on a different instance. For propagations, if the owner instance for a queue table containing a destination queue becomes unavailable, then queue ownership is transferred automatically to another instance in the cluster. A queue-to-queue propagation to a buffered destination queue uses a service to provide transparent failover in a RAC environment. That is, a **propagation job** for a queue-to-queue propagation automatically connects to the instance that owns the destination queue.

The service used by a queue-to-queue propagation always runs on the owner instance of the destination queue. This service is created only for buffered queues in a RAC database. If you plan to use buffered messaging with a RAC database, then messages can be enqueued into a buffered queue on any instance. If messages are enqueued on an instance that does not own the queue, then the messages are sent to the correct instance, but it is more efficient to enqueue messages on the instance that owns the queue. The service can be used to connect to the owner instance of the queue before enqueuing messages into a buffered queue.

Queue-to-dblink propagations do not use services. To make the propagation job connect to the correct instance on the **destination database**, manually reconfigure the

database link from the **source database** to connect to the instance that owns the destination queue.

The NAME column in the DBA_SERVICES data dictionary view contains the service name for a queue. The NETWORK_NAME column in the DBA_QUEUES data dictionary view contains the network name for a queue. Do not manage the services for queue-to-queue propagations in any way. Oracle manages them automatically. For queue-to-dblink propagations, use the network name as the service name in the connect string of the database link to connect to the correct instance.

The DBA_QUEUE_TABLES data dictionary view contains information about the owner instance for a queue table. A queue table can contain multiple queues. In this case, each queue in a queue table has the same owner instance as the queue table.

Note: If a queue contains or will contain **captured message**s in a RAC environment, then queue-to-queue propagations should be used to propagate messages to a RAC destination database. If a queue-to-dblink propagation propagates captured messages to a RAC destination database, then this propagation must use an instance-specific database link that refers to the owner instance of the destination queue. If such a propagation connects to any other instance, then the propagation will raise an error.

See Also:

- "Queue-to-Queue Propagations" on page 3-5
- "Streams Capture Processes and Oracle Real Application Clusters" on page 2-21
- "Streams Apply Processes and Oracle Real Application Clusters" on page 4-9
- "Buffered Queues" on page 3-21
- *Oracle Database Reference* for more information about the DBA_QUEUE_TABLES data dictionary view
- *Oracle Streams Advanced Queuing User's Guide and Reference* for more information about queues and RAC
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the ALTER_QUEUE_TABLE procedure

Commit-Time Queues

You can control the order in which **user-enqueued message**s in a **queue** are browsed or dequeued. Message ordering in a queue is determined by its **queue table**, and you can specify message ordering for a queue table during queue table creation. Specifically, the sort_list parameter in the DBMS_AQADM.CREATE_QUEUE_TABLE procedure determines how user-enqueued messages are ordered. Oracle Database 10*g* Release 2 introduces **commit-time queue**s. Each message in a commit-time queue is ordered by an **approximate commit system change number (approximate CSCN)** which is obtained when the transaction that enqueued the message commits.

Commit-time ordering is specified for a queue table, and queues that use the queue table are called commit-time queues. When commit_time is specified for the sort_list parameter in the DBMS_AQADM.CREATE_QUEUE_TABLE procedure, the resulting queue table uses commit-time ordering.

For Oracle Database 10*g* Release 2, the default `sort_list` setting for queue tables created by the `SET_UP_QUEUE` procedure in the `DBMS_STREAMS_ADM` package is `commit_time`. For releases prior to Oracle Database 10*g* Release 2, the default is `enq_time`, which is described in the section that follows. When the `queue_table` parameter in the `SET_UP_QUEUE` procedure specifies an existing queue table, message ordering in the queue created by `SET_UP_QUEUE` is determined by the existing queue table.

When to Use Commit-Time Queues

A user or application can share information by enqueuing messages into a **queue** in an Oracle database. The enqueued messages can be shared within a single database or propagated to other databases, and the messages can be LCRs or **user message**s. For example, messages can be enqueued when an application-specific message occurs or when a trigger is fired for a database change. Also, in a heterogeneous environment, an application can enqueue messages that originated at a non-Oracle database into a queue in an Oracle database.

Other than `commit_time`, the settings for the `sort_list` parameter in the `CREATE_QUEUE_TABLE` procedure are `priority` and `enq_time`. The `priority` setting orders messages by the priority specified during enqueue, highest priority to lowest priority. The `enq_time` setting orders messages by the time when they were enqueued, oldest to newest.

Commit-time queues are useful when an environment must support either of the following requirements for concurrent enqueues of **user-enqueued message**s:

- Transactional Dependency Ordering During Dequeue
- Consistent Browse of Messages in a Queue

Commit-time queues support these requirements. Neither priority nor enqueue time ordering supports these requirements because both allow transactional dependency violations and nonconsistent browses. Both settings allow transactional dependency violations, because messages are dequeued independent of the original dependencies. Also, both settings allow nonconsistent browses of the messages in a queue, because multiple browses performed without any dequeue operations between them can result in different sets of messages.

See Also:

- "Introduction to Message Staging and Propagation" on page 3-1
- "Message Propagation Between Queues" on page 3-4
- *Oracle Streams Replication Administrator's Guide* for more information about **heterogeneous information sharing**

Transactional Dependency Ordering During Dequeue

A transactional dependency occurs when one database transaction requires that another database transaction commits before it can commit successfully. Messages that contain information about database transactions can be enqueued into a queue. For example, a database trigger can fire to enqueue messages. Figure 3–4 shows how enqueue time ordering does not support transactional dependency ordering during dequeue of such messages.

Figure 3–4 Transactional Dependency Violation During Dequeue

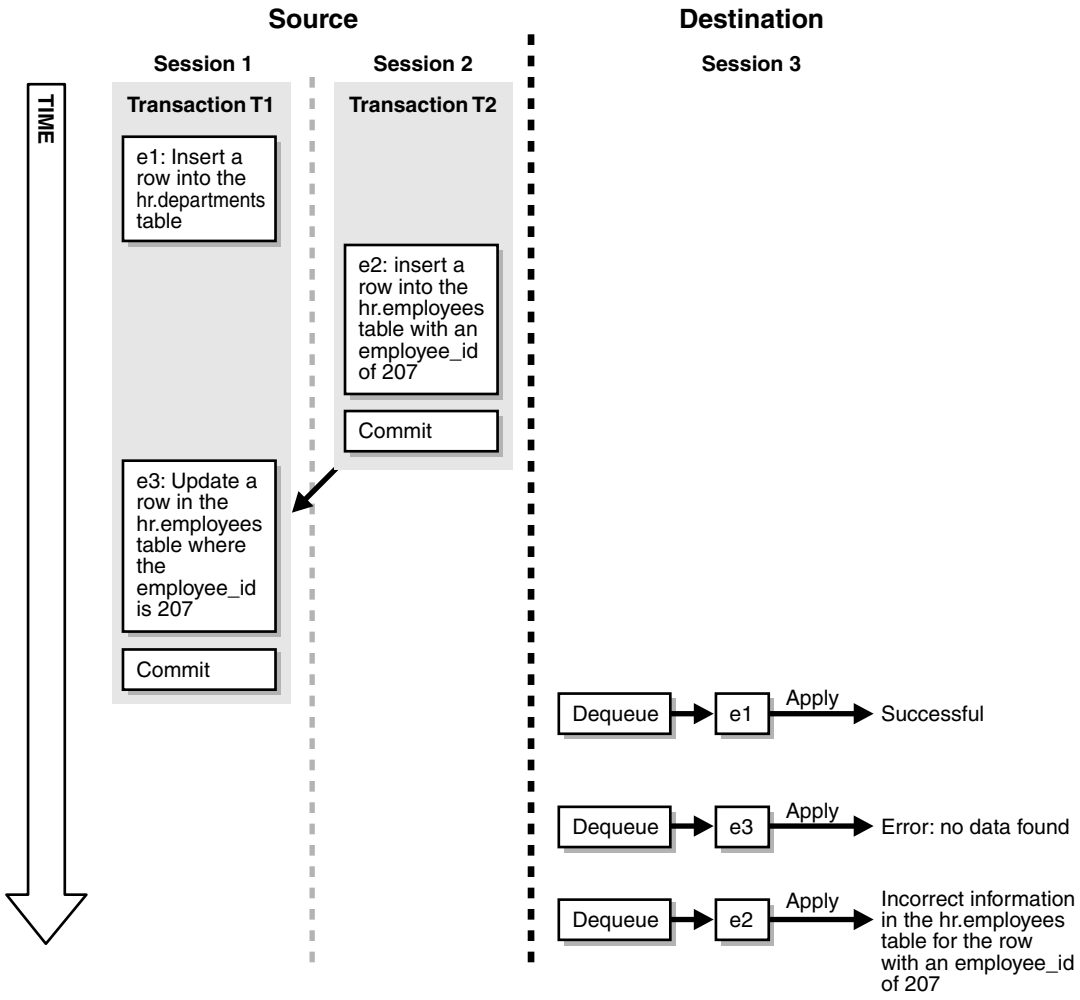


Figure 3–4 shows how transactional dependency ordering can be violated with enqueue time ordering. The transaction that enqueued message `e2` was committed before the transaction that enqueued messages `e1` and `e3` was committed, and the update in message `e3` depends on the insert in message `e2`. So, the correct dequeue order that supports transactional dependencies is `e2`, `e1`, `e3`. However, with enqueue time ordering, `e3` can be dequeued before `e2`. Therefore, when `e3` is dequeued, an error results when an application attempts to apply the change in `e3` to the `hr.employees` table. Also, after all three messages are dequeued, a row in the `hr.employees` table contains the wrong information because the change in `e3` was not executed.

Consistent Browse of Messages in a Queue

Figure 3–5 shows how enqueue time ordering does not support consistent browse of messages in a queue.

Figure 3–5 Inconsistent Browse of Messages in a Queue

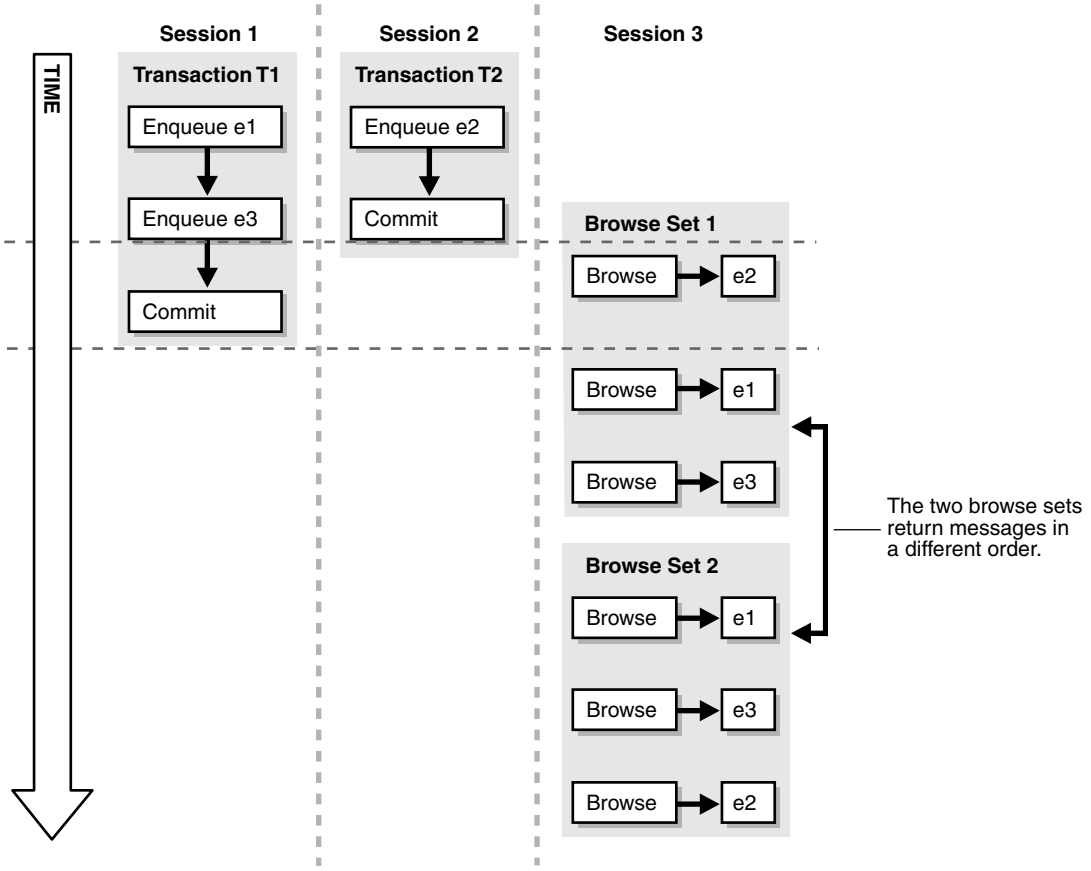


Figure 3–5 shows that a client browsing messages in a queue is not guaranteed a definite order with enqueue time ordering. Sessions 1 and 2 are concurrent sessions that are enqueuing messages. Session 3 shows two sets of client browses that return the three enqueued messages in different orders. If the client requires deterministic ordering of messages, then the client might fail. For example, the client might perform a browse to initiate a program state, and a subsequent dequeue might return messages in a different order than expected.

How Commit-Time Queues Work

The commit system change number (CSCN) for a message that is enqueued into a **queue** is not known until the redo record for the commit of the transaction that includes the message is written to the redo log. The CSCN cannot be recorded when the message is enqueued. Commit-time queues use the current SCN of the database when a transaction is committed as the approximate CSCN for all of the messages in the transaction. The order of messages in a commit-time queue is based on the approximate CSCN of the transaction that enqueued the messages.

In a commit-time queue, messages in a transaction are not visible to dequeue and browse operations until a deterministic order for the messages can be established using the approximate CSCN. When multiple transactions are enqueuing messages concurrently into the same commit-time queue, two or more transactions can commit at nearly the same time, and the commit intervals for these transactions can overlap. In this case, the messages in these transactions are not visible until all of the transactions have committed. At that time, the order of the messages can be determined using the

approximate CSCN of each transaction. Dependencies are maintained by using the approximate CSCN for messages rather than the enqueue time. Read consistency for browses is maintained by ensuring that only messages with a fully determined order are visible.

A commit-time queue always maintains transactional dependency ordering for messages that are based on database transactions. However, applications and users can enqueue messages that are not based on database transactions. For these messages, if dependencies exist between transactions, then the application or user must ensure that transactions are committed in the correct order and that the commit intervals of the dependent transactions do not overlap.

The approximate CSCNs of transactions recorded by a commit-time queue might not reflect the actual commit order of these transactions. For example, transaction 1 and transaction 2 can commit at nearly the same time after enqueuing their messages. The approximate CSCN for transaction 1 can be lower than the approximate CSCN for transaction 2, but transaction 1 can take more time to complete the commit than transaction 2. In this case, the actual CSCN for transaction 2 is lower than the actual CSCN for transaction 1.

Note: The `sort_list` parameter in `CREATE_QUEUE_TABLE` can be set to the following:

```
priority, commit_time
```

In this case, ordering is done by priority first and commit time second. Therefore, this setting does not ensure transactional dependency ordering and browse read consistency for messages with different priorities. However, transactional dependency ordering and browse read consistency are ensured for messages with the same priority.

See Also: "Creating an ANYDATA Queue" on page 12-2 for information about creating a commit-time queue

Streams Staging and Propagation Architecture

This section describes **buffered queue**s, **propagation job**s, and **secure queue**s, and how they are used in Streams. In addition, this section discusses how **transactional queue**s handle **captured message**s and **user-enqueued message**s, as well as the need for a **Streams data dictionary** at databases that propagate **captured message**s.

This section contains the following topics:

- Streams Pool
- Buffered Queues
- Propagation Jobs
- Secure Queues
- Transactional and Nontransactional Queues
- Streams Data Dictionary for Propagations

See Also:

- *Oracle Streams Advanced Queuing User's Guide and Reference* for more information about AQ infrastructure
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_JOB` package

Streams Pool

The **Streams pool** is a portion of memory in the System Global Area (SGA) that is used by Streams. The Streams pool stores **buffered queue message**s in memory, and it provides memory for **capture process**es and **apply process**es. The Streams pool always stores LCRs captured by a capture process, and it stores LCRs and messages that are enqueued into a buffered queue by applications or users.

The Streams pool is initialized the first time any one of the following actions occur in a database:

- A message is enqueued into a buffered queue. Data Pump export and import operations initialize the Streams pool because these operations use buffered queues.
- A capture process is started.
- An apply process is started.

The size of the Streams pool is determined in one of the following ways:

- Streams Pool Size Set by Automatic Shared Memory Management
- Streams Pool Size Set Manually by a Database Administrator
- Streams Pool Size Set by Default

Note: If the Streams pool cannot be initialized, then an `ORA-00832` error is returned. If this happens, then first ensure that there is enough space in the SGA for the Streams pool. If necessary, reset the `SGA_MAX_SIZE` initialization parameter to increase the SGA size. Next, either set the `SGA_TARGET` or the `STREAMS_POOL_SIZE` initialization parameter (or both).

Streams Pool Size Set by Automatic Shared Memory Management

The Automatic Shared Memory Management feature manages the size of the Streams pool when the `SGA_TARGET` initialization parameter is set to a nonzero value. If the `STREAMS_POOL_SIZE` initialization parameter also is set to a nonzero value, then Automatic Shared Memory Management uses this value as a minimum for the Streams pool. You can set a minimum size if your environment needs a minimum amount of memory in the Streams pool to function properly.

See Also: *Oracle Database Administrator's Guide* and *Oracle Database Reference* for more information about Automatic Shared Memory Management and the `SGA_TARGET` initialization parameter

Streams Pool Size Set Manually by a Database Administrator

If the `STREAMS_POOL_SIZE` initialization parameter is set to a nonzero value, and the `SGA_TARGET` parameter is set to `0` (zero), then the Streams pool size is the value specified by the `STREAMS_POOL_SIZE` parameter, in bytes. If you plan to set the

Streams pool size manually, then you can use the V$STREAMS_POOL_ADVICE dynamic performance view to determine an appropriate setting for the STREAMS_POOL_SIZE initialization parameter.

See Also: "Monitoring the Streams Pool" on page 26-3

Streams Pool Size Set by Default

If both the STREAMS_POOL_SIZE and the SGA_TARGET initialization parameters are set to 0 (zero), then, by default, the first use of Streams in a database transfers an amount of memory equal to 10% of the shared pool from the buffer cache to the Streams pool. The buffer cache is set by the DB_CACHE_SIZE initialization parameter, and the shared pool size is set by the SHARED_POOL_SIZE initialization parameter.

For example, consider the following configuration in a database before Streams is used for the first time:

- DB_CACHE_SIZE is set to 100 MB.
- SHARED_POOL_SIZE is set to 80 MB.
- STREAMS_POOL_SIZE is set to zero.
- SGA_TARGET is set to zero.

Given this configuration, the amount of memory allocated after Streams is used for the first time is the following:

- The buffer cache has 92 MB.
- The shared pool has 80 MB.
- The Streams pool has 8 MB.

The first use of Streams in a database is the first attempt to allocate memory from the Streams pool. Memory is allocated from the Streams pool in the following ways:

- A message is enqueued into a buffered queue. The message can be an LCR captured by a capture process, or it can be a user-enqueued LCR or message.
- A capture process is started.
- An apply process is started.

See Also:

- "Setting Initialization Parameters Relevant to Streams" on page 10-4 for more information about the STREAMS_POOL_SIZE initialization parameter
- "Multiple Capture Processes in a Single Database" on page 2-25
- "Buffered Queues" on page 3-21
- "Multiple Apply Processes in a Single Database" on page 4-16

Buffered Queues

A **buffered queue** includes the following storage areas:

- **Streams pool** memory associated with a queue that contains **message**s that were captured by a **capture process** or enqueued by applications or users
- Part of a **queue table** that stores messages that have spilled from memory to disk

Queue tables are stored on disk. Buffered queues enable Oracle to optimize messages by buffering them in the SGA instead of always storing them in a queue table.

If the size of the Streams pool is not managed automatically, then you should increase the size of the Streams pool by 10 MB for each buffered queue in a database. Buffered queues improve performance, but some of the information in a buffered queue can be lost if the instance containing the buffered queue shuts down normally or abnormally. Streams automatically recovers from these cases, assuming full database recovery is performed on the instance.

Messages in a buffered queue can spill from memory into the queue table if they have been staged in the buffered queue for a period of time without being dequeued, or if there is not enough space in memory to hold all of the messages. Messages that spill from memory are stored in the appropriate `AQ$_queue_table_name_p` table, where *`queue_table_name`* is the name of the queue table for the queue. Also, for each spilled message, information is stored in the `AQ$_queue_table_name_d` table about any **propagation**s and **apply process**es that are eligible for processing the message.

Captured messages are always stored in a buffered queue, but user-enqueued LCRs and user-enqueued non-LCR messages might or might not be stored in a buffered queue. For a **user-enqueued message**, the enqueue operation specifies whether the enqueued message is stored in the buffered queue or in the persistent queue. A persistent queue only stores messages on hard disk in a queue table, not in memory. The `delivery_mode` attribute in the `enqueue_options` parameter of the `DBMS_AQ.ENQUEUE` procedure determines whether a message is stored in the buffered queue or the persistent queue. Specifically, if the `delivery_mode` attribute is the default `PERSISTENT`, then the message is enqueued into the persistent queue. If it is set to `BUFFERED`, then the message is enqueued as the buffered queue. When a transaction is moved to the error queue, all messages in the transaction always are stored in a queue table, not in a buffered queue.

Note:

- Using triggers on queue tables is not recommended because it can have a negative impact on performance. Also, the use of triggers on index-organized queue tables is not supported.
- Although buffered and persistent messages can be stored in the same queue, it is sometimes more convenient to think of a queue having a buffered portion and a persistent portion, referred to here as "buffered queue" and "persistent queue".

See Also:

- "Streams Pool" on page 3-19
- *Oracle Streams Advanced Queuing User's Guide and Reference* for detailed conceptual information about buffered messaging and for information about using buffered messaging

Propagation Jobs

A Streams **propagation** is configured internally using the DBMS_JOB package. Therefore, a **propagation job** is a job used by a propagation that propagates **message**s from a **source queue** to a **destination queue**. Like other jobs configured using the DBMS_JOB package, propagation jobs have an owner, and they use job queue processes (*Jnnn*) as needed to execute jobs.

The following procedures can create a propagation job when they create a propagation:

- The ADD_GLOBAL_PROPAGATION_RULES procedure in the DBMS_STREAMS_ADM package
- The ADD_SCHEMA_PROPAGATION_RULES procedure in the DBMS_STREAMS_ADM package
- The ADD_TABLE_PROPAGATION_RULES procedure in the DBMS_STREAMS_ADM package
- The ADD_SUBSET_PROPAGATION_RULE procedure in the DBMS_STREAMS_ADM package
- The CREATE_PROPAGATION procedure in the DBMS_PROPAGATION_ADM package

When one of these procedures creates a propagation, a new propagation job is created in the following cases:

- If the queue_to_queue parameter is set to true, then a new propagation job always is created for the propagation. Each queue-to-queue propagation has its own propagation job. However, a job queue process can be used by multiple propagation jobs.
- If the queue_to_queue parameter is set to false, then a propagation job is created when no propagation job exists for the source queue and database link specified. If a propagation job already exists for the specified source queue and database link, then the new propagation uses the existing propagation job and shares this propagation job with all of the other queue-to-dblink propagations that use the same database link.

A propagation job for a queue-to-dblink propagation can be used by more than one propagation. All destination queues at a database receive messages from a single source queue through a single propagation job. By using a single propagation job for multiple destination queues, Streams ensures that a message is sent to a **destination database** only once, even if the same message is received by multiple destination queues in the same database. Communication resources are conserved because messages are not sent more than once to the same database.

Note: The source queue owner performs the propagation, but the propagation job is owned by the user who creates it. These two users might or might not be the same.

See Also: "Queue-to-Queue Propagations" on page 3-5

Propagation Scheduling and Streams Propagations

A **propagation schedule** specifies how often a propagation job propagates messages from a source queue to a destination queue. Each queue-to-queue propagation has its own propagation job and propagation schedule, but queue-to-dblink propagations that use the same propagation job have the same propagation schedule.

A default propagation schedule is established when a new propagation job is created by a procedure in the DBMS_STREAMS_ADM or DBMS_PROPAGATION_ADM package.

The default schedule has the following properties:

- The start time is SYSDATE().
- The duration is NULL, which means infinite.
- The next time is NULL, which means that propagation restarts as soon as it finishes the current duration.
- The latency is three seconds, which is the wait time after a queue becomes empty to resubmit the propagation job. Therefore, the latency is the maximum wait, in seconds, in the propagation window for a message to be propagated after it is enqueued.

You can alter the schedule for a propagation job using the ALTER_PROPAGATION_SCHEDULE procedure in the DBMS_AQADM package. Changes made to a propagation job affect all propagations that use the propagation job.

See Also:

- "Propagation Jobs" on page 3-22
- "Altering the Schedule of a Propagation Job" on page 12-10

Propagation Jobs and RESTRICTED SESSION

When the restricted session is enabled during system startup by issuing a STARTUP RESTRICT statement, propagation jobs with enabled propagation schedules do not propagate messages. When the restricted session is disabled, each propagation schedule that is enabled and ready to run will run when there is an available job queue process.

When the restricted session is enabled in a running database by the SQL statement ALTER SYSTEM ENABLE RESTRICTED SESSION, any running propagation job continues to run to completion. However, any new propagation job submitted for a propagation schedule is not started. Therefore, propagation for an enabled schedule can eventually come to a halt.

Secure Queues

Secure queues are **queue**s for which AQ agents must be associated explicitly with one or more database users who can perform queue operations, such as enqueue and dequeue. The owner of a secure queue can perform all queue operations on the queue, but other users cannot perform queue operations on a secure queue, unless they are configured as **secure queue users**. In Streams, secure queues can be used to ensure that only the appropriate users and **Streams client**s enqueue **message**s into a queue and dequeue messages from a queue.

Secure Queues and the SET_UP_QUEUE Procedure

All ANYDATA queues created using the SET_UP_QUEUE procedure in the DBMS_STREAMS_ADM package are secure queues. When you use the SET_UP_QUEUE procedure to create a queue, any user specified by the queue_user parameter is configured as a secure queue user of the queue automatically, if possible. The queue user is also granted ENQUEUE and DEQUEUE privileges on the queue. To enqueue messages into and dequeue messages from a queue, a queue user must also have EXECUTE privilege on the DBMS_STREAMS_MESSAGING package or the DBMS_AQ package. The SET_UP_QUEUE procedure does not grant either of these privileges.

Also, a message cannot be enqueued into a queue unless a subscriber who can dequeue the message is configured.

To configure a queue user as a secure queue user, the SET_UP_QUEUE procedure creates an AQ agent with the same name as the user name, if one does not already exist. The user must use this agent to perform queue operations on the queue. If an agent with this name already exists and is associated with the queue user only, then the existing agent is used. SET_UP_QUEUE then runs the ENABLE_DB_ACCESS procedure in the DBMS_AQADM package, specifying the agent and the user.

If you use the SET_UP_QUEUE procedure in the DBMS_STREAMS_ADM package to create a secure queue, and you want a user who is not the queue owner and who was not specified by the queue_user parameter to perform operations on the queue, then you can configure the user as a secure queue user of the queue manually. Alternatively, you can run the SET_UP_QUEUE procedure again and specify a different queue_user for the queue. In this case, SET_UP_QUEUE skips queue creation, but it configures the user specified by queue_user as a secure queue user of the queue.

If you create an ANYDATA queue using the DBMS_AQADM package, then you use the secure parameter when you run the CREATE_QUEUE_TABLE procedure to specify whether the queue is secure or not. The queue is secure if you specify true for the secure parameter when you run this procedure. When you use the DBMS_AQADM package to create a secure queue, and you want to allow users to perform queue operations on the secure queue, then you must configure these secure queue users manually.

Secure Queues and Streams Clients

When you create a **capture process** or an **apply process**, an AQ agent of the secure queue associated with the Streams process is configured automatically, and the user who runs the Streams process is specified as a secure queue user for this queue automatically. Therefore, a capture process is configured to enqueue into its secure queue automatically, and an apply process is configured to dequeue from its secure queue automatically. In either case, the AQ agent has the same name as the Streams client.

For a capture process, the user specified as the capture_user is the user who runs the capture process. For an apply process, the user specified as the apply_user is the user who runs the apply process. If no capture_user or apply_user is specified, then the user who invokes the procedure that creates the Streams process is the user who runs the Streams process.

Also, if you change the capture_user for a capture process or the apply_user for an apply process, then the specified capture_user or apply_user is configured as a secure queue user of the queue used by the Streams process. However, the old **capture user** or **apply user** remains configured as a secure queue user of the queue. To remove the old user, run the DISABLE_DB_ACCESS procedure in the DBMS_AQADM package, specifying the old user and the relevant AQ agent. You might also want to drop the agent if it is no longer needed. You can view the AQ agents and their associated users by querying the DBA_AQ_AGENT_PRIVS data dictionary view.

When you create a **messaging client**, an AQ agent of the secure queue with the same name as the messaging client is associated with the user who runs the procedure that creates the messaging client. This messaging client user is specified as a secure queue user for this queue automatically. Therefore, this user can use the messaging client to dequeue messages from the queue.

A capture process, an apply process, or a messaging client can be associated with only one user. However, one user can be associated with multiple Streams clients, including

multiple capture processes, apply processes, and messaging clients. For example, an apply process cannot have both hr and oe as apply users, but hr can be the apply user for multiple apply processes.

If you drop a capture process, apply process, or messaging client, then the users who were configured as secure queue users for these Streams clients remain secure queue users of the queue. To remove these users as secure queue users, run the DISABLE_DB_ACCESS procedure in the DBMS_AQADM package for each user. You might also want to drop the agent if it is no longer needed.

Note: No configuration is necessary for **propagation**s and secure queues. Therefore, when a propagation is dropped, no additional steps are necessary to remove secure queue users from the propagation's queues.

See Also:

- "Enabling a User to Perform Operations on a Secure Queue" on page 12-3
- "Disabling a User from Performing Operations on a Secure Queue" on page 12-5
- *Oracle Database PL/SQL Packages and Types Reference* for more information about AQ agents and using the DBMS_AQADM package

Transactional and Nontransactional Queues

A **transactional queue** is a **queue** in which **user-enqueued message**s can be grouped into a set that are applied as one transaction. That is, an **apply process** performs a COMMIT after it applies all the user-enqueued messages in a group. The SET_UP_QUEUE procedure in the DBMS_STREAMS_ADM package always creates a transactional queue.

A **nontransactional queue** is one in which each user-enqueued message is its own transaction. That is, an apply process performs a COMMIT after each user-enqueued message it applies. In either case, the user-enqueued messages might or might not contain user-created LCRs.

The difference between transactional and nontransactional queues is important only for user-enqueued messages. An apply process always applies **captured message**s in transactions that preserve the transactions executed at the **source database**. Table 3–1 shows apply process behavior for each type of message and each type of queue.

Table 3–1 Apply Process Behavior for Transactional and Nontransactional Queues

Message Type	Transactional Queue	Nontransactional Queue
Captured Messages	Apply process preserves the original transaction	Apply process preserves the original transaction
User-Enqueued Messages	Apply a user-specified group of user-enqueued messages as one transaction	Apply each user-enqueued message in its own transaction

See Also:

- "Managing ANYDATA Queues" on page 12-1
- *Oracle Streams Advanced Queuing User's Guide and Reference* for more information about message grouping

Streams Data Dictionary for Propagations

When a database object is prepared for **instantiation** at a **source database**, a Streams data dictionary is populated automatically at the database where changes to the object are captured by a **capture process**. The Streams data dictionary is a multiversioned copy of some of the information in the primary data dictionary at a source database. The Streams data dictionary maps object numbers, object version information, and internal column numbers from the source database into table names, column names, and column datatypes. This mapping keeps each **captured message** as small as possible, because the message can store numbers rather than names internally.

The mapping information in the Streams data dictionary at the source database is needed to evaluate **rule**s at any database that propagates the captured messages from the source database. To make this mapping information available to a **propagation**, Oracle automatically populates a multiversioned Streams data dictionary at each database that has a Streams propagation. Oracle automatically sends internal messages that contain relevant information from the Streams data dictionary at the source database to all other databases that receive captured messages from the source database.

The Streams data dictionary information contained in these internal messages in a **queue** might or might not be propagated by a propagation. Which Streams data dictionary information to propagate depends on the **rule set**s for the propagation. When a propagation encounters Streams data dictionary information for a table, the propagation rule sets are evaluated with partial information that includes the source database name, table name, and table owner. If the partial rule evaluation of these rule sets determines that there might be relevant LCRs for the given table from the specified database, then the Streams data dictionary information for the table is propagated.

When Streams data dictionary information is propagated to a destination queue, it is incorporated into the Streams data dictionary at the database that contains the destination queue, in addition to being enqueued into the destination queue. Therefore, a propagation reading the destination queue in a directed networks configuration can forward LCRs immediately without waiting for the Streams data dictionary to be populated. In this way, the Streams data dictionary for a source database always reflects the correct state of the relevant database objects for the LCRs relating to these database objects.

See Also:

- "The Streams Data Dictionary" on page 2-37
- Chapter 6, "How Rules Are Used in Streams"

4

Streams Apply Process

This chapter explains the concepts and architecture of the Streams **apply process**.

This chapter contains these topics:

- Introduction to the Apply Process
- Apply Process Rules
- Message Processing with an Apply Process
- Datatypes Applied
- Streams Apply Processes and RESTRICTED SESSION
- Streams Apply Processes and Oracle Real Application Clusters
- Apply Process Architecture

See Also: Chapter 13, "Managing an Apply Process"

Introduction to the Apply Process

An **apply process** is an optional Oracle background process that dequeues **message**s from a specific **queue**. These messages can be logical change records (**LCR**s) or **user message**s, and an apply process either applies each message directly or passes it as a parameter to an apply handler. An **apply handler** is a user-defined procedure used by an apply process for customized processing of messages. The LCRs dequeued by an apply process contain the results of data manipulation language (DML) changes or data definition language (DDL) changes that an apply process can apply to database objects in a **destination database**. A **user-enqueued message** dequeued by an apply process is of type `ANYDATA` and can contain any message, including an LCR or a **user message**.

Note: An apply process can only dequeue messages from an `ANYDATA` queue, not a **typed queue**.

Apply Process Rules

An **apply process** applies changes based on **rule**s that you define. Each **rule** specifies the database objects and types of changes for which the rule evaluates to `TRUE`. You can place these rules in the **positive rule set** or **negative rule set** for the apply process.

If a rule evaluates to `TRUE` for a change, and the rule is in the positive rule set for an apply process, then the apply process applies the change. If a rule evaluates to `TRUE` for a change, and the rule is in the negative rule set for an apply process, then the

apply process discards the change. If an apply process has both a positive and a negative rule set, then the negative rule set is always evaluated first.

You can specify apply process rules for LCRs at the following levels:

- A table rule applies or discards either row changes resulting from DML changes or DDL changes to a particular table. Subset rules are table rules that include a subset of the row changes to a particular table.
- A schema rule applies or discards either row changes resulting from DML changes or DDL changes to the database objects in a particular schema.
- A global rule applies or discards either all row changes resulting from DML changes or all DDL changes in the **queue** associated with an apply process.

For non-LCR **message**s, you can create rules to control apply process behavior for specific types of messages.

See Also:

- Chapter 5, "Rules"
- Chapter 6, "How Rules Are Used in Streams"

Message Processing with an Apply Process

An **apply process** is a flexible mechanism for processing the **message**s in a **queue**. You have options to consider when you configure one or more apply processes for your environment. The following sections discuss the types of messages that an apply process can apply and the ways in which it can apply them.

- Processing Captured or User-Enqueued Messages with an Apply Process
- Message Processing Options for an Apply Process

Processing Captured or User-Enqueued Messages with an Apply Process

A single apply process can dequeue either of the following types of messages:

- **Captured message:** A **message** that was captured implicitly by a **capture process**. A captured message contains a **logical change record (LCR)**.
- **User-enqueued message:** A message that was enqueued explicitly by an application, a user, or an apply process. A user-enqueued message can contain either an LCR or a **user message**.

A single apply process cannot dequeue both captured and user-enqueued messages. If a **queue** at a **destination database** contains both captured and user-enqueued messages, then the destination database must have at least two apply processes to process the messages.

A single apply process can apply user-enqueued messages that originated at multiple databases. However, a single apply process can apply captured messages from only one **source database**, because processing these LCRs requires knowledge of the dependencies, meaningful transaction ordering, and transactional boundaries at the **source database**. For a captured message, the source database is the database where the change encapsulated in the LCR was generated in the redo log.

Captured messages from multiple databases can be sent to a single **destination queue**. However, if a single queue contains captured messages from multiple source databases, then there must be multiple apply processes retrieving these LCRs. Each of these apply processes should be configured to receive captured messages from exactly

one source database using **rule**s. Oracle recommends that you use a separate ANYDATA queue for captured messages from each source database.

Also, each apply process can apply captured messages from only one capture process. If multiple capture processes are running on a source database, and LCRs from more than one of these capture processes are applied at a destination database, then there must be one apply process to apply changes from each capture process. In such an environment, Oracle recommends that each ANYDATA queue used by a capture process, propagation, or apply process have captured messages from at most one capture process from a particular source database. A queue can contain LCRs from more than one capture process if each capture process is capturing changes that originated at a different source database.

See Also:

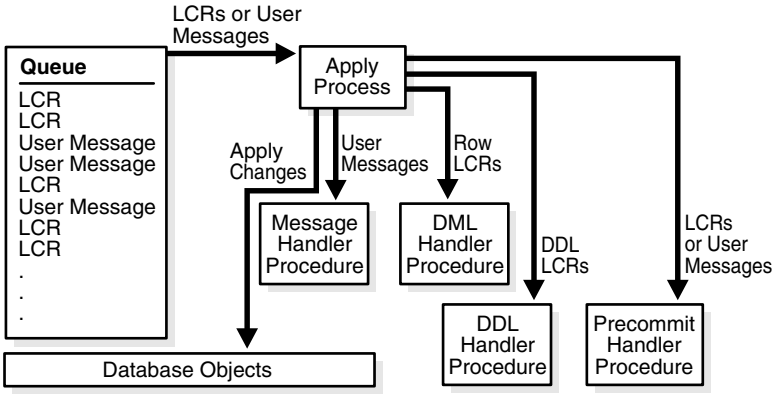
- "Introduction to Message Staging and Propagation" on page 3-1 for more information about captured and user-enqueued messages
- "Creating an Apply Process" on page 13-2 for information about creating an apply process to apply captured or user-enqueued messages

Message Processing Options for an Apply Process

Your options for **message** processing depend on whether or not the message received by an apply process is an LCR.

Figure 4–1 shows the message processing options for an apply process.

Figure 4–1 Apply Process Message Processing Options



The following sections describe these message processing options:

- LCR Processing
- Non-LCR User Message Processing
- Audit Commit Information for Messages Using Precommit Handlers
- Considerations for Apply Handlers
- Summary of Message Processing Options

LCR Processing

You can configure an apply process to process each LCR that it dequeues in the following ways:

- Apply the LCR Directly
- Call a User Procedure to Process the LCR

Apply the LCR Directly If you use this option, then an apply process applies the LCR without running a user procedure. The apply process either successfully applies the change in the LCR to a database object or, if a **conflict** or an apply error is encountered, tries to resolve the error with a conflict handler or a user-specified procedure called an **error handler**.

If a conflict handler can resolve the conflict, then it either applies the LCR or it discards the change in the LCR. If the error handler can resolve the error, then it should apply the LCR, if appropriate. An error handler can resolve an error by modifying the LCR before applying it. If the conflict handler or error handler cannot resolve the error, then the apply process places the transaction, and all LCRs associated with the transaction, into the error queue.

Call a User Procedure to Process the LCR If you use this option, then an apply process passes the LCR as a parameter to a user procedure for processing. The user procedure can process the LCR in a customized way.

A user procedure that processes row LCRs resulting from DML statements is called a **DML handler**. A user procedure that processes DDL LCRs resulting from DDL statements is called a **DDL handler**. An apply process can have many DML handlers but only one DDL handler, which processes all DDL LCRs dequeued by the apply process.

For each table associated with an apply process, you can set a separate DML handler to process each of the following types of operations in row LCRs:

- `INSERT`
- `UPDATE`
- `DELETE`
- `LOB_UPDATE`

For example, the `hr.employees` table can have one DML handler procedure to process `INSERT` operations and a different DML handler procedure to process `UPDATE` operations. Alternatively, the `hr.employees` table can use the same DML handler procedure for each type of operation.

A user procedure can be used for any customized processing of LCRs. For example, if you want each insert into a particular table at the **source database** to result in inserts into multiple tables at the **destination database**, then you can create a user procedure that processes `INSERT` operations on the table to accomplish this. Or, if you want to log DDL changes before applying them, then you can create a user procedure that processes DDL operations to accomplish this.

A DML handler should never commit and never roll back, except to a named savepoint that the user procedure has established. To execute a row LCR inside a DML handler, invoke the `EXECUTE` member procedure for the row LCR. To execute a DDL LCR inside a DDL handler, invoke the `EXECUTE` member procedure for the DDL LCR.

To set a DML handler, use the SET_DML_HANDLER procedure in the DBMS_APPLY_ADM package. You can either set a DML handler for a specific apply process, or you can set a DML handler to be a general DML handler that is used by all apply processes in the database. If a DML handler for an operation on a table is set for a specific apply process, and another DML handler is a general handler for the same operation on the same table, then the specific DML handler takes precedence over the general DML handler.

To associate a DDL handler with a particular apply process, use the ddl_handler parameter in the CREATE_APPLY or the ALTER_APPLY procedure in the DBMS_APPLY_ADM package.

You create an error handler in the same way that you create a DML handler, except that you set the error_handler parameter to true when you run the SET_DML_HANDLER procedure. An error handler is invoked only if an apply error results when an apply process tries to apply a row LCR for the specified operation on the specified table.

Typically, DML handlers and DDL handlers are used in Streams **replication** environments to perform custom processing of LCRs, but these handlers can be used in nonreplication environments as well. For example, such handlers can be used to record changes made to database objects without replicating these changes.

Attention: Do not modify LONG, LONG RAW, or nonassembled LOB column data in an LCR with DML handlers, error handlers, or **custom rule-based transformation** functions. DML handlers and error handlers can modify LOB columns in row LCRs that have been constructed by **LOB assembly**.

Note: When you run the SET_DML_HANDLER procedure, you specify the object for which the handler is used. This object does not need to exist at the destination database.

See Also:

- "Logical Change Records (LCRs)" on page 2-2 for more information about row LCRs and DDL LCRs
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the EXECUTE member procedure for LCR types
- Chapter 7, "Rule-Based Transformations"
- *Oracle Streams Replication Administrator's Guide* for more information about DML handlers and DDL handlers

Non-LCR User Message Processing

A **user-enqueued message** that does not contain an LCR is processed by the **message handler** specified for an apply process. A message handler is a user-defined procedure that can process **user message**s in a customized way for your environment.

The message handler offers advantages in any environment that has applications that need to update one or more remote databases or perform some other remote action. These applications can enqueue user messages into a **queue** at the local database, and Streams can propagate each user message to the appropriate queues at destination

databases. If there are multiple destinations, then Streams provides the infrastructure for automatic propagation and processing of these messages at these destinations. If there is only one destination, then Streams still provides a layer between the application at the source database and the application at the destination database, so that, if the application at the remote database becomes unavailable, then the application at the source database can continue to function normally.

For example, a message handler can convert a user message into an electronic mail message. In this case, the user message can contain the attributes you would expect in an electronic mail message, such as `from`, `to`, `subject`, `text_of_message`, and so on. After converting a message into an electronic mail messages, the message handler can send it out through an electronic mail gateway.

You can specify a message handler for an apply process using the `message_handler` parameter in the `CREATE_APPLY` or the `ALTER_APPLY` procedure in the `DBMS_APPLY_ADM` package. A Streams apply process always assumes that a non-LCR message has no dependencies on any other messages in the queue. If parallelism is greater than `1` for an apply process that applies user-enqueued messages, then these messages can be dequeued by a message handler in any order. Therefore, if dependencies exist between these messages in your environment, then Oracle recommends that you set apply process parallelism to `1`.

> **See Also:** "Managing the Message Handler for an Apply Process" on page 13-12

Audit Commit Information for Messages Using Precommit Handlers

You can use a **precommit handler** to audit commit directives for **captured message**s and transaction boundaries for user-enqueued messages. A precommit handler is a user-defined PL/SQL procedure that can receive the commit information for a transaction and process the commit information in any customized way. A precommit handler can work with a DML handler or a message handler.

For example, a handler can improve performance by caching data for the length of a transaction. This data can include cursors, temporary LOBs, data from a message, and so on. The precommit handler can release or execute the objects cached by the handler when a transaction completes.

A precommit handler executes when the apply process commits a transaction. You can use the `commit_serialization` apply process parameter to control the commit order for an apply process.

Commit Directives for Captured Messages When you are using a **capture process**, and a user commits a transaction, the capture process captures an internal commit directive for the transaction if the transaction contains row LCRs that were captured. Once enqueued into a queue, these commit directives can be propagated to **destination queue**s, along with the LCRs in a transaction. A precommit handler receives the commit SCN for these internal commit directives in the queue of an apply process before they are processed by the apply process.

Transaction Boundaries for User-Enqueued Messages A user or application can enqueue messages into a queue and then issue a `COMMIT` statement to end the transaction. The enqueued messages are organized into a message group. Once enqueued into a queue, the messages in a message group can be propagated to other queues. When an apply process is configured to process user-enqueued messages, it generates a single transaction identifier and commit SCN for all the messages in a message group. Transaction identifiers and commit SCN values generated by an individual apply process have no relation to the source transaction, or to the values generated by any

other apply process. A precommit handler configured for such an apply process receives the commit SCN supplied by the apply process.

> **See Also:** "Managing the Precommit Handler for an Apply Process" on page 13-13

Considerations for Apply Handlers

The following are considerations for using apply handlers:

- DML handlers, DDL handlers, and message handlers can execute an LCR by calling the LCR's `EXECUTE` member procedure.
- All applied DDL LCRs commit automatically. Therefore, if a DDL handler calls the `EXECUTE` member procedure of a DDL LCR, then a commit is performed automatically.
- If necessary, an apply handler can set a Streams session **tag**.
- An apply handler can call a Java stored procedure that is published (or wrapped) in a PL/SQL procedure.
- If an apply process tries to invoke an apply handler that does not exist or is invalid, then the apply process aborts.
- If an apply handler invokes a procedure or function in an Oracle-supplied package, then the user who runs the apply handler must have direct `EXECUTE` privilege on the package. It is not sufficient to grant this privilege through a role.

> **See Also:**
>
> - *Oracle Database PL/SQL Packages and Types Reference* for more information about the `EXECUTE` member procedure for LCR types
> - *Oracle Streams Replication Administrator's Guide* for more information about Streams tags

Summary of Message Processing Options

Table 4–1 summarizes the message processing options available when you are using one or more of the **apply handler**s described in the previous sections. Apply handlers are optional for row LCRs and DDL LCRs because an apply process can apply these messages directly. However, a message handler is required for processing user messages. In addition, an apply process dequeues a message only if the message satisfies the **rule set**s for the apply process. In general, a message satisfies the rule sets for an apply process if *no rules* in the **negative rule set** evaluate to `TRUE` for the message, and *at least one rule* in the **positive rule set** evaluates to `TRUE` for the message.

Table 4–1 Summary of Message Processing Options

Apply Handler	Type of Message	Default Apply Process Behavior	Scope of User Procedure
DML Handler or Error Handler	Row LCR	Execute DML	One operation on one table
DDL Handler	DDL LCR	Execute DDL	Entire apply process
Message Handler	User Message	Create error transaction (if no message handler exists)	Entire apply process
Precommit Handler	Commit directive for transactions that include row LCRs or user messages	Commit transaction	Entire apply process

In addition to the message processing options described in this section, you can use the `SET_ENQUEUE_DESTINATION` procedure in the `DBMS_APPLY_ADM` package to instruct an apply process to enqueue messages into a specified destination queue. Also, you can control message execution using the `SET_EXECUTE` procedure in the `DBMS_APPLY_ADM` package.

See Also:

- Chapter 6, "How Rules Are Used in Streams" for more information about rule sets for **Streams client**s and for information about how messages satisfy rule sets
- "Specifying Message Enqueues by Apply Processes" on page 13-15
- "Specifying Execute Directives for Apply Processes" on page 13-16

Datatypes Applied

When applying row LCRs resulting from DML changes to tables, an **apply process** applies changes made to columns of the following datatypes:

- `VARCHAR2`
- `NVARCHAR2`
- `NUMBER`
- `LONG`
- `DATE`
- `BINARY_FLOAT`
- `BINARY_DOUBLE`
- `TIMESTAMP`
- `TIMESTAMP WITH TIME ZONE`
- `TIMESTAMP WITH LOCAL TIME ZONE`
- `INTERVAL YEAR TO MONTH`
- `INTERVAL DAY TO SECOND`

- RAW
- LONG RAW
- CHAR
- NCHAR
- CLOB
- NCLOB
- BLOB
- UROWID

An apply process does not apply row LCRs containing the results of DML changes in columns of the following datatypes: BFILE, ROWID, and user-defined type (including object types, REFs, varrays, nested tables, and Oracle-supplied types). Also, an apply process cannot apply changes to columns if the columns have been encrypted using transparent data encryption. An apply process raises an error if it attempts to apply a row LCR that contains information about a column of an unsupported datatype. Next, the apply process moves the transaction that includes the LCR into the error queue.

See Also:

- "Datatypes Captured" on page 2-6
- *Oracle Database SQL Reference* for more information about these datatypes

Streams Apply Processes and RESTRICTED SESSION

When restricted session is enabled during system startup by issuing a STARTUP RESTRICT statement, **apply process**es do not start, even if they were running when the database shut down. When the restricted session is disabled, each apply process that was not stopped is started.

When restricted session is enabled in a running database by the SQL statement ALTER SYSTEM ENABLE RESTRICTED SESSION, it does not affect any running apply processes. These apply processes continue to run and apply messages. If a stopped apply process is started in a restricted session, then the apply process does not actually start until the restricted session is disabled.

Streams Apply Processes and Oracle Real Application Clusters

You can configure a Streams **apply process** to apply changes in an Oracle Real Application Clusters (RAC) environment. Each apply process is started and stopped on the owner instance for its ANYDATA queue, even if the start or stop procedure is run on a different instance.

If the owner instance for a **queue table** containing a **queue** used by an apply process becomes unavailable, then queue ownership is transferred automatically to another instance in the cluster. Also, an apply process will follow its queue to a different instance if the current owner instance becomes unavailable. The queue itself follows the rules for primary instance and secondary instance ownership. In addition, if the apply process was enabled when the owner instance became unavailable, then the apply process is restarted automatically on the new owner instance. If the apply process was disabled when the owner instance became unavailable, then the apply process remains disabled on the new owner instance.

The DBA_QUEUE_TABLES data dictionary view contains information about the owner instance for a queue table. Also, in a RAC environment, an apply **coordinator process**, its corresponding apply **reader server**, and all of its **apply servers** run on a single instance.

See Also:

- "Queues and Oracle Real Application Clusters" on page 3-13 for information about primary and secondary instance ownership for queues
- "Streams Capture Processes and Oracle Real Application Clusters" on page 2-21
- *Oracle Database Reference* for more information about the DBA_QUEUE_TABLES data dictionary view
- "Persistent Apply Process Status upon Database Restart" on page 4-16

Apply Process Architecture

You can create, alter, start, stop, and drop an **apply process**, and you can define apply process **rule**s that control which **message**s an apply process dequeues from its **queue**. Messages are applied in the security domain of the **apply user** for an apply process. The apply user dequeues all messages that satisfy the apply process **rule set**s. The apply user can apply messages directly to database objects. In addition, the apply user runs all **custom rule-based transformation**s specified by the rules in these rule sets. The apply user also runs user-defined **apply handler**s.

The apply user must have the necessary privileges to apply changes, including EXECUTE privilege on the rule sets used by the apply process, EXECUTE privilege on all custom rule-based transformation functions specified for rules in the **positive rule set**, EXECUTE privilege on any apply handlers, and privileges to dequeue messages from the apply process queue. An apply process can be associated with only one user, but one user can be associated with many apply processes.

See Also: "Configuring a Streams Administrator" on page 10-1 for information about the required privileges

This section discusses the following topics:

- Apply Process Components
- Apply Process Creation
- Streams Data Dictionary for an Apply Process
- Apply Process Parameters
- Persistent Apply Process Status upon Database Restart
- The Error Queue

Apply Process Components

An **apply process** consists of the following components:

- A **reader server** that dequeues **message**s. The reader server is a parallel execution server that computes dependencies between LCRs and assembles messages into transactions. The reader server then returns the assembled transactions to the coordinator process, which assigns them to idle apply servers.
- A **coordinator process** that gets transactions from the reader server and passes them to apply servers. The coordinator process name is `a`*`nnn`*, where *nnn* is a coordinator process number. Valid coordinator process names include `a001` through `a999`. The coordinator process is an Oracle background process.
- One or more **apply servers** that apply LCRs to database objects as DML or DDL statements or that pass the LCRs to their appropriate **apply handler**s. For non-LCR messages, the apply servers pass the messages to the **message handler**. Apply servers can also enqueue LCR and non-LCR messages into a **queue** specified by the `DBMS_APPLY_ADM.SET_ENQUEUE_DESTINATION` procedure. Each apply server is a parallel execution server. If an apply server encounters an error, then it then tries to resolve the error with a user-specified **conflict** handler or error handler. If an apply server cannot resolve an error, then it rolls back the transaction and places the entire transaction, including all of its messages, in the error queue.

  When an apply server commits a completed transaction, this transaction has been applied. When an apply server places a transaction in the error queue and commits, this transaction also has been applied.

If a transaction being handled by an apply server has a dependency on another transaction that is not known to have been applied, then the apply server contacts the coordinator process and waits for instructions. The coordinator process monitors all of the apply servers to ensure that transactions are applied and committed in the correct order.

> **See Also:** *Oracle Streams Replication Administrator's Guide* for more information about apply processes and dependencies

Reader Server States

The state of a reader server describes what the reader server is doing currently. You can view the state of the reader server for an apply process by querying the `V$STREAMS_APPLY_READER` dynamic performance view. The following reader server states are possible:

- `INITIALIZING` - Starting up
- `IDLE` - Performing no work
- `DEQUEUE MESSAGES` - Dequeuing messages from the apply process queue
- `SCHEDULE MESSAGES` - Computing dependencies between messages and assembling messages into transactions
- `SPILLING` - Spilling unapplied messages from memory to hard disk
- `PAUSED` - Waiting for a DDL LCR to be applied

> **See Also:** "Displaying Information About the Reader Server for Each Apply Process" on page 22-6 for a query that displays the state of an apply process reader server

Coordinator Process States

The state of a coordinator process describes what the coordinator process is doing currently. You can view the state of a coordinator process by querying the V$STREAMS_APPLY_COORDINATOR dynamic performance view. The following coordinator process states are possible:

- INITIALIZING - Starting up
- APPLYING - Passing transactions to apply servers
- SHUTTING DOWN CLEANLY - Stopping without an error
- ABORTING - Stopping because of an apply error

> **See Also:** "Displaying General Information About Each Coordinator Process" on page 22-9 for a query that displays the state of a coordinator process

Apply Server States

The state of an apply server describes what the apply server is doing currently. You can view the state of each apply server for an apply process by querying the V$STREAMS_APPLY_SERVER dynamic performance view. The following apply server states are possible:

- INITIALIZING - Starting up.
- IDLE - Performing no work.
- RECORD LOW-WATERMARK - Performing an administrative action that maintains information about the apply progress, which is used in the ALL_APPLY_PROGRESS and DBA_APPLY_PROGRESS data dictionary views.
- ADD PARTITION - Performing an administrative action that adds a partition that is used for recording information about in-progress transactions.
- DROP PARTITION - Performing an administrative action that drops a partition that was used to record information about in-progress transactions.
- EXECUTE TRANSACTION - Applying a transaction.
- WAIT COMMIT - Waiting to commit a transaction until all other transactions with a lower commit SCN are applied. This state is possible only if the COMMIT_SERIALIZATION apply process parameter is set to a value other than none and the PARALELLISM apply process parameter is set to a value greater than 1.
- WAIT DEPENDENCY - Waiting to apply an LCR in a transaction until another transaction, on which it has a dependency, is applied. This state is possible only if the PARALELLISM apply process parameter is set to a value greater than 1.
- WAIT FOR NEXT CHUNK - Waiting for the next set of LCRs for a large transaction.
- TRANSACTION CLEANUP - Cleaning up an applied transaction, which includes removing LCRs from the apply process queue.

> **See Also:** "Displaying Information About the Apply Servers for Each Apply Process" on page 22-12 for a query that displays the state of each apply process apply server

Apply Process Creation

You can create an **apply process** using the DBMS_STREAMS_ADM package or the DBMS_APPLY_ADM package. Using the DBMS_STREAMS_ADM package to create an apply process is simpler because defaults are used automatically for some configuration options. In addition, when you use the DBMS_STREAMS_ADM package, a **rule set** is created for the apply process and **rule**s can be added to the rule set automatically. The rule set is a **positive rule set** if the inclusion_rule parameter is set to true (the default). It is a **negative rule set** if the inclusion_rule parameter is set to false.

Alternatively, using the DBMS_APPLY_ADM package to create an apply process is more flexible, and you create one or more rule sets and rules for the apply process either before or after it is created.

An apply process created by the procedures in the DBMS_STREAMS_ADM package can apply **message**s only at the local database. To create an apply process that applies messages at a remote database, use the CREATE_APPLY procedure in the DBMS_APPLY_ADM package.

Changes applied by an apply process created by the DBMS_STREAMS_ADM package generate tags in the redo log at the **destination database** with a value of 00 (double zero), but you can set the **tag** value if you use the CREATE_APPLY procedure. Alternatively, you can set the tag using the ALTER_APPLY procedure in the DBMS_APPLY_ADM package.

When you create an apply process by running the CREATE_APPLY procedure in the DBMS_APPLY_ADM package, you can specify nondefault values for the apply_captured, apply_database_link, and apply_tag parameters. Then you can use the procedures in the DBMS_STREAMS_ADM package or the DBMS_RULE_ADM package to add rules to a rule set for the apply process.

If you create more than one apply process in a database, then the apply processes are completely independent of each other. These apply processes do not synchronize with each other, even if they apply LCRs from the same **source database**.

See Also:

- "Creating an Apply Process" on page 13-2
- *Oracle Streams Replication Administrator's Guide* for more information about Streams tags

Streams Data Dictionary for an Apply Process

When a database object is prepared for **instantiation** at a **source database**, a Streams data dictionary is populated automatically at the database where changes to the object are captured by a **capture process**. The Streams data dictionary is a multiversioned copy of some of the information in the primary data dictionary at a source database. The Streams data dictionary maps object numbers, object version information, and internal column numbers from the source database into table names, column names, and column datatypes. This mapping keeps each **captured message** as small as possible because a captured message can often use numbers rather than names internally.

Unless a captured message is passed as a parameter to a **custom rule-based transformation** during capture or propagation, the mapping information in the Streams data dictionary at the source database is needed to interpret the contents of the LCR at any database that applies the captured message. To make this mapping information available to an **apply process**, Oracle automatically populates a

multiversioned Streams data dictionary at each **destination database** that has a Streams apply process. Oracle automatically propagates relevant information from the Streams data dictionary at the source database to all other databases that apply captured messages from the source database.

See Also:

- "The Streams Data Dictionary" on page 2-37
- "Streams Data Dictionary for Propagations" on page 3-26

Apply Process Parameters

After creation, an **apply process** is disabled so that you can set the apply process parameters for your environment before starting the process for the first time. Apply process parameters control the way an apply process operates. For example, the `time_limit` apply process parameter specifies the amount of time an apply process runs before it is shut down automatically. After you set the apply process parameters, you can start the apply process.

See Also:

- "Setting an Apply Process Parameter" on page 13-11
- This section does not discuss all of the available apply process parameters. See the `DBMS_APPLY_ADM.SET_PARAMETER` procedure in the *Oracle Database PL/SQL Packages and Types Reference* for detailed information about all of the apply process parameters.

Apply Process Parallelism

The `parallelism` apply process parameter specifies the number of **apply servers** that can concurrently apply transactions. For example, if `parallelism` is set to `5`, then an apply process uses a total of five apply servers. The **reader server** is a parallel execution server. So, if `parallelism` is set to `5`, then an apply process uses a total of six parallel execution servers, assuming six parallel execution servers are available in the database. An apply process always uses two or more parallel execution servers.

Note:

- Resetting the `parallelism` parameter automatically stops and restarts the apply process when the currently executing transactions are applied. This operation can take some time depending on the size of the transactions.
- Setting the `parallelism` parameter to a number higher than the number of available parallel execution servers can disable the apply process. Make sure the `PROCESSES` and `PARALLEL_MAX_SERVERS` initialization parameters are set appropriately when you set the `parallelism` apply process parameter.

See Also:

- "Apply Process Components" on page 4-11 for more information about apply servers and the reader server
- *Oracle Database Administrator's Guide* for information about managing parallel execution servers

Commit Serialization

Apply servers can apply nondependent transactions at the **destination database** in an order that is different from the commit order at the **source database**. Dependent transactions are always applied at the destination database in the same order as they were committed at the source database.

You control whether the **apply servers** can apply nondependent transactions in a different order at the destination database using the `commit_serialization` apply parameter. This parameter has the following settings:

- `full`: An apply process always commits all transactions in the order in which they were committed at the source database. This setting is the default.
- `none`: An apply process can commit nondependent transactions in any order. An apply process always commits dependent transactions in the order in which they were committed at the source database. Performance is best if you specify this value.

If you specify `none`, then it is possible that a destination database commits changes in a different order than the source database. For example, suppose two nondependent transactions are committed at the source database in the following order:

1. Transaction A
2. Transaction B

At the destination database, these transactions might be committed in the opposite order:

1. Transaction B
2. Transaction A

Automatic Restart of an Apply Process

You can configure an apply process to stop automatically when it reaches certain predefined limits. The `time_limit` apply process parameter specifies the amount of time an apply process runs, and the `transaction_limit` apply process parameter specifies the number of transactions an apply process can apply. The apply process stops automatically when it reaches these limits.

The `disable_on_limit` parameter controls whether an apply process becomes disabled or restarts when it reaches a limit. If you set the `disable_on_limit` parameter to `y`, then the apply process is disabled when it reaches a limit and does not restart until you restart it explicitly. If, however, you set the `disable_on_limit` parameter to `n`, then the apply process stops and restarts automatically when it reaches a limit.

When an apply process is restarted, it gets a new session identifier, and the parallel execution servers associated with the apply process also get new session identifiers. However, the **coordinator process** number (a*nnn*) remains the same.

Stop or Continue on Error

Using the `disable_on_error` apply process parameter, you can instruct an apply process to become disabled when it encounters an error or to continue applying transactions after it encounters an error.

See Also: "The Error Queue" on page 4-16

Multiple Apply Processes in a Single Database

If you run multiple **apply process**es in a single database, consider increasing the size of the System Global Area (SGA). In a Real Application Clusters environment, consider increasing the size of the SGA for each instance. Use the SGA_MAX_SIZE initialization parameter to increase the SGA size. Also, if the size of the **Streams pool** is not managed automatically in the database, then you should increase the size of the Streams pool by 1 MB for each apply process parallelism. For example, if you have two apply processes running in a database, and the parallelism parameter is set to 4 for one of them and 1 for the other, then increase the Streams pool by 5 MB (4 + 1 = 5 parallelism).

Note: The size of the Streams pool is managed automatically if the SGA_TARGET initialization parameter is set to a nonzero value.

See Also:

- Streams Pool on page 3-19
- "Setting Initialization Parameters Relevant to Streams" on page 10-4 for more information about the STREAMS_POOL_SIZE initialization parameter

Persistent Apply Process Status upon Database Restart

An **apply process** maintains a persistent status when the database running the apply process is shut down and restarted. For example, if an apply process is enabled when the database is shut down, then the apply process automatically starts when the database is restarted. Similarly, if an apply process is disabled or aborted when a database is shut down, then the apply process is not started and retains the disabled or aborted status when the database is restarted.

The Error Queue

The error queue contains all of the current apply errors for a database. If there are multiple **apply process**es in a database, then the error queue contains the apply errors for each apply process. To view information about apply errors, query the DBA_APPLY_ERROR data dictionary view or use Enterprise Manager.

The error queue stores information about transactions that could not be applied successfully by the apply processes running in a database. A transaction can include many **message**s. When an unhandled error occurs during apply, an apply process automatically moves all of the messages in the transaction that satisfy the apply process **rule set**s to the error queue.

You can correct the condition that caused an error and then reexecute the transaction that caused the error. For example, you might modify a row in a table to correct the condition that caused an error.

When the condition that caused the error has been corrected, you can either reexecute the transaction in the error queue using the EXECUTE_ERROR or EXECUTE_ALL_ERRORS procedure, or you can delete the transaction from the error queue using the DELETE_ERROR or DELETE_ALL_ERRORS procedure. These procedures are in the DBMS_APPLY_ADM package.

When you reexecute a transaction in the error queue, you can specify that the transaction be executed either by the user who originally placed the error in the error queue or by the user who is reexecuting the transaction. Also, the current Streams **tag** for the apply process is used when you reexecute a transaction in the error queue.

A reexecuted transaction uses any relevant **apply handler**s and **conflict resolution** handlers. If, to resolve the error, a row LCR in an error queue must be modified before it is executed, then you can configure a **DML handler** to process the row LCR that caused the error in the error queue. In this case, the DML handler can modify the row LCR in some way to avoid a repetition of the same error. The row LCR is passed to the DML handler when you reexecute the error containing the row LCR.

The error queue contains information about errors encountered at the local **destination database** only. It does not contain information about errors for apply processes running in other databases in a Streams environment.

The error queue uses the **exception queue**s in the database. When you create an `ANYDATA` queue using the `SET_UP_QUEUE` procedure in the `DBMS_STREAMS_ADM` package, the procedure creates a **queue table** for the queue if one does not already exist. When a queue table is created, an exception queue is created automatically for the queue table. Multiple queues can use a single queue table, and each queue table has one exception queue. Therefore, a single exception queue can store errors for multiple queues and multiple apply processes.

An exception queue only contains the apply errors for its queue table, but the Streams error queue contains information about all of the apply errors in each exception queue in a database. You should use the procedures in the `DBMS_APPLY_ADM` package to manage Streams apply errors. You should not dequeue apply errors from an exception queue directly.

Note: If a **messaging client** encounters an error when it is dequeuing messages, then the messaging client moves these messages to the exception queue associated with the its queue table. However, information about messaging client errors is not stored in the error queue. Only information about apply process errors is stored in the error queue.

See Also:

- "Managing Apply Errors" on page 13-23
- "Checking for Apply Errors" on page 22-15
- "Displaying Detailed Information About Apply Errors" on page 22-16
- "Managing an Error Handler" on page 13-18
- Chapter 6, "How Rules Are Used in Streams" for more information about rule sets for **Streams client**s and for information about how messages satisfy rule sets
- *Oracle Database PL/SQL Packages and Types Reference* for more information on the `DBMS_APPLY_ADM` package
- *Oracle Database Reference* for more information about the `DBA_APPLY_ERROR` data dictionary view

5
Rules

This chapter explains the concepts related to rules.

This chapter contains these topics:

- The Components of a Rule
- Rule Set Evaluation
- Database Objects and Privileges Related to Rules

See Also:

- Chapter 6, "How Rules Are Used in Streams"
- Chapter 14, "Managing Rules"
- Chapter 28, "Rule-Based Application Example"

The Components of a Rule

A **rule** is a database object that enables a client to perform an action when an event occurs and a condition is satisfied. A rule consists of the following components:

- Rule Condition
- Rule Evaluation Context (optional)
- Rule Action Context (optional)

Each rule is specified as a condition that is similar to the condition in the `WHERE` clause of a SQL query. You can group related rules together into **rule sets**. A single rule can be in one rule set, multiple rule sets, or no rule sets.

Rule sets are evaluated by a **rules engine**, which is a built-in part of Oracle. Both user-created applications and Oracle features, such as Streams, can be clients of the rules engine.

Note: A rule must be in a rule set for it to be evaluated.

Rule Condition

A **rule condition** combines one or more **expression**s and conditions and returns a Boolean value, which is a value of TRUE, FALSE, or NULL (unknown). An **expression** is a combination of one or more values and operators that evaluate to a value. A value can be data in a table, data in variables, or data returned by a SQL function or a PL/SQL function. For example, the following expression includes only a single value:

```
salary
```

The following expression includes two values (salary and .1) and an operator (*):

```
salary * .1
```

The following condition consists of two expressions (salary and 3800) and a condition (=):

```
salary = 3800
```

This logical condition evaluates to TRUE for a given row when the salary column is 3800. Here, the value is data in the salary column of a table.

A single rule condition can include more than one condition combined with the AND, OR, and NOT logical conditions to a form compound condition. A logical condition combines the results of two component conditions to produce a single result based on them or to invert the result of a single condition. For example, consider the following compound condition:

```
salary = 3800 OR job_title = 'Programmer'
```

This rule condition contains two conditions joined by the OR logical condition. If either condition evaluates to TRUE, then the rule condition evaluates to TRUE. If the logical condition were AND instead of OR, then both conditions must evaluate to TRUE for the entire rule condition to evaluate to TRUE.

Variables in Rule Conditions

Rule conditions can contain variables. When you use variables in rule conditions, precede each variable with a colon (:). The following is an example of a variable used in a rule condition:

```
:x = 55
```

Variables let you refer to data that is not stored in a table. A variable can also improve performance by replacing a commonly occurring expression. Performance can improve because, instead of evaluating the same expression multiple times, the variable is evaluated once.

A rule condition can also contain an evaluation of a call to a subprogram. Such a condition is evaluated in the same way as other conditions. That is, it evaluates to a value of TRUE, FALSE, or NULL (unknown). The following is an example of a condition that contains a call to a simple function named is_manager that determines whether an employee is a manager:

```
is_manager(employee_id) = 'Y'
```

Here, the value of employee_id is determined by data in a table where employee_id is a column.

You can use user-defined types for variables. Therefore, variables can have attributes. When a variable has attributes, each attribute contains partial data for the variable. In

rule conditions, you specify attributes using dot notation. For example, the following condition evaluates to TRUE if the value of attribute z in variable y is 9:

```
:y.z = 9
```

Note: A rule cannot have a NULL (or empty) rule condition.

See Also:

- *Oracle Database SQL Reference* for more information about conditions, expressions, and operators
- *Oracle Database Application Developer's Guide - Object-Relational Features* for more information about user-defined types

Simple Rule Conditions

A simple rule condition is a condition that has one of the following forms:

- *simple_rule_expression condition constant*
- *constant condition simple_rule_expression*
- *constant condition constant*

Simple Rule Expressions In a simple rule condition, a *simple_rule_expression* is one of the following:

- Table column.
- Variable.
- Variable attribute.
- Method result where the method either takes no arguments or constant arguments and the method result can be returned by the variable method function, so that the expression is one of the datatypes supported for simple rules. Such methods include LCR member subprograms that meet these requirements, such as GET_TAG, GET_VALUE, GET_COMPATIBLE, GET_EXTRA_ATTRIBUTE, and so on.

For table columns, variables, variable attributes, and method results, the following datatypes can be used in simple rule conditions:

- VARCHAR2
- NVARCHAR2
- NUMBER
- DATE
- BINARY_FLOAT
- BINARY_DOUBLE
- TIMESTAMP
- TIMESTAMP WITH TIME ZONE
- TIMESTAMP WITH LOCAL TIME ZONE
- RAW
- CHAR

Use of other datatypes in expressions results in nonsimple rule conditions.

Conditions In a simple rule condition, a *condition* is one of the following:

- <=
- <
- =
- >
- >=
- !=
- IS NULL
- IS NOT NULL

Use of other conditions results in nonsimple rule conditions.

Constants A *constant* is a fixed value. A constant can be:

- A number, such as 12 or 5.4
- A character, such as x or $
- A character string, such as "this is a string"

Examples of Simple Rule Conditions The following conditions are simple rule conditions, assuming the datatypes used in expressions are supported in simple rule conditions:

- tab1.col = 5
- tab2.col != 5
- :v1 > 'aaa'
- :v2.a1 < 10.01
- :v3.m() = 10
- :v4 IS NOT NULL
- 1 = 1
- 'abc' > 'AB'
- :date_var < to_date('04-01-2004, 14:20:17', 'mm-dd-yyyy, hh24:mi:ss')
- :adt_var.ts_attribute >= to_timestamp('04-01-2004, 14:20:17 PST', 'mm-dd-yyyy, hh24:mi:ss TZR')
- :my_var.my_to_upper('abc') = 'ABC'

Rules with simple rule conditions are called simple rules. You can combine two or more simple conditions with the logical conditions AND and OR for a rule, and the rule remains simple. For example, rules with the following conditions are simple rules:

- tab1.col = 5 AND :v1 > 'aaa'
- tab1.col = 5 OR :v1 > 'aaa'

However, using the NOT logical condition in a rule condition causes the rule to be nonsimple.

Benefits of Simple Rules Simple rules are important for the following reasons:

- Simple rules are indexed by the **rules engine** internally.
- Simple rules can be evaluated without executing SQL.
- Simple rules can be evaluated with partial data.

When a client uses the `DBMS_RULE.EVALUATE` procedure to evaluate an event, the client can specify that only simple rules should be evaluated by specifying `true` for the `simple_rules_only` parameter.

See Also:

- *Oracle Database SQL Reference* for more information about conditions and logical conditions
- *Oracle Database PL/SQL Packages and Types Reference* for more information about LCR types and their member subprograms

Rule Evaluation Context

An **evaluation context** is a database object that defines external data that can be referenced in **rule condition**s. The external data can exist as variables, table data, or both. The following analogy might be helpful: If the rule condition were the `WHERE` clause in a SQL query, then the external data in the evaluation context would be the tables and bind variables referenced in the `FROM` clause of the query. That is, the **expression**s in the rule condition should reference the tables, table aliases, and variables in the evaluation context to make a valid `WHERE` clause.

A rule evaluation context provides the necessary information for interpreting and evaluating the rule conditions that reference external data. For example, if a rule refers to a variable, then the information in the rule evaluation context must contain the variable type. Or, if a rule refers to a table alias, then the information in the evaluation context must define the table alias.

The objects referenced by a rule are determined by the rule evaluation context associated with it. The rule owner must have the necessary privileges to access these objects, such as `SELECT` privilege on tables, `EXECUTE` privilege on types, and so on. The rule condition is resolved in the schema that owns the evaluation context.

For example, consider a rule evaluation context named `hr_evaluation_context` that contains the following information:

- Table alias `dep` corresponds to the `hr.departments` table.
- Variables `loc_id1` and `loc_id2` are both of type `NUMBER`.

The `hr_evaluation_context` rule evaluation context provides the necessary information for evaluating the following rule condition:

```
dep.location_id IN (:loc_id1, :loc_id2)
```

In this case, the rule condition evaluates to `TRUE` for a row in the `hr.departments` table if that row has a value in the `location_id` column that corresponds to either of the values passed in by the `loc_id1` or `loc_id2` variables. The rule cannot be interpreted or evaluated properly without the information in the `hr_evaluation_context` rule evaluation context. Also, notice that dot notation is used to specify the column `location_id` in the `dep` table alias.

Note: Views are not supported as base tables in evaluation contexts.

Explicit and Implicit Variables

The value of a variable referenced in a rule condition can be explicitly specified when the rule is evaluated, or the value of a variable can be implicitly available given the event.

Explicit variables are supplied by the caller at evaluation time. These values are specified by the `variable_values` parameter when the `DBMS_RULE.EVALUATE` procedure is run.

Implicit variables are not given a value supplied by the caller at evaluation time. The value of an implicit variable is obtained by calling the variable value function. You define this function when you specify the `variable_types` list during the creation of an evaluation context using the `CREATE_EVALUATION_CONTEXT` procedure in the `DBMS_RULE_ADM` package. If the value for an implicit variable is specified during evaluation, then the specified value overrides the value returned by the variable value function.

Specifically, the `variable_types` list is of type `SYS.RE$VARIABLE_TYPE_LIST`, which is a list of variables of type `SYS.RE$VARIABLE_TYPE`. Within each instance of `SYS.RE$VARIABLE_TYPE` in the list, the function used to determine the value of an implicit variable is specified as the `variable_value_function` attribute.

Whether variables are explicit or implicit is the choice of the designer of the application using the **rules engine**. The following are reasons for using an implicit variable:

- The caller of the `DBMS_RULE.EVALUATE` procedure does not need to know anything about the variable, which can reduce the complexity of the application using the rules engine. For example, a variable can call a function that returns a value based on the data being evaluated.
- The caller might not have `EXECUTE` privileges on the variable value function.
- The caller of the `DBMS_RULE.EVALUATE` procedure does not know the variable value based on the event, which can improve security if the variable value contains confidential information.
- The variable will be used infrequently, and the variable's value always can be derived if necessary. Making such variables implicit means that the caller of the `DBMS_RULE.EVALUATE` procedure does not need to specify many uncommon variables.

For example, in the following rule condition, the values of variable `x` and variable `y` could be specified explicitly, but the value of the variable `max` could be returned by running the `max` function:

```
:x = 4 AND :y < :max
```

Alternatively, variable `x` and `y` could be implicit variables, and variable `max` could be an explicit variable. So, there is no syntactic difference between explicit and implicit variables in the rule condition. You can determine whether a variable is explicit or implicit by querying the `DBA_EVALUATION_CONTEXT_VARS` data dictionary view. For explicit variables, the `VARIABLE_VALUE_FUNCTION` field is `NULL`. For implicit variables, this field contains the name of the function called by the implicit variable.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for more information about the DBMS_RULE and DBMS_RULE_ADM packages, and for more information about the Oracle-supplied rule types
- *Oracle Database Reference* for more information about the DBA_EVALUATION_CONTEXT_VARS data dictionary view

Evaluation Context Association with Rule Sets and Rules

To be evaluated, each rule must be associated with an evaluation context or must be part of a **rule set** that is associated with an evaluation context. A single evaluation context can be associated with multiple rules or rule sets. The following list describes which evaluation context is used when a rule is evaluated:

- If an evaluation context is associated with a rule, then it is used for the rule whenever the rule is evaluated, and any evaluation context associated with the rule set being evaluated is ignored.
- If a rule does not have an evaluation context, but an evaluation context was specified for the rule when it was added to a rule set using the ADD_RULE procedure in the DBMS_RULE_ADM package, then the evaluation context specified in the ADD_RULE procedure is used for the rule when the rule set is evaluated.
- If no rule evaluation context is associated with a rule and none was specified by the ADD_RULE procedure, then the evaluation context of the rule set is used for the rule when the rule set is evaluated.

Note: If a rule does not have an evaluation context, and you try to add it to a rule set that does not have an evaluation context, then an error is raised, unless you specify an evaluation context when you run the ADD_RULE procedure.

Evaluation Function

You have the option of creating an evaluation function to be run with a rule evaluation context. You can use an evaluation function for the following reasons:

- You want to bypass the rules engine and instead evaluate events using the evaluation function.
- You want to filter events so that some events are evaluated by the evaluation function and other events are evaluated by the rules engine.

You associate a function with a rule evaluation context by specifying the function name for the evaluation_function parameter when you create the rule evaluation context with the CREATE_EVALUATION_CONTEXT procedure in the DBMS_RULE_ADM package. The rules engine invokes the evaluation function during the evaluation of any rule set that uses the evaluation context.

The DBMS_RULE.EVALUATE procedure is overloaded. The function must have each parameter in one of the DBMS_RULE.EVALUATE procedures, and the type of each parameter must be same as the type of the corresponding parameter in the DBMS_RULE.EVALUATE procedure, but the names of the parameters can be different.

An evaluation function has the following return values:

- DBMS_RULE_ADM.EVALUATION_SUCCESS: The user specified evaluation function completed the rule set evaluation successfully. The rules engine returns the results of the evaluation obtained by the evaluation function to the rules engine client using the DBMS_RULE.EVALUATE procedure.
- DBMS_RULE_ADM.EVALUATION_CONTINUE: The rules engine evaluates the rule set as if there were no evaluation function. The evaluation function is not used, and any results returned by the evaluation function are ignored.
- DBMS_RULE_ADM.EVALUATION_FAILURE: The user-specified evaluation function failed. Rule set evaluation stops, and an error is raised.

If you always want to bypass the rules engine, then the evaluation function should return either EVALUATION_SUCCESS or EVALUATION_FAILURE. However, if you want to filter events so that some events are evaluated by the evaluation function and other events are evaluated by the rules engine, then the evaluation function can return all three return values, and it returns EVALUATION_CONTINUE when the rules engine should be used for evaluation.

If you specify an evaluation function for an evaluation context, then the evaluation function is run during evaluation when the evaluation context is used by a rule set or rule.

> **See Also:** *Oracle Database PL/SQL Packages and Types Reference* for more information about the evaluation function specified in the DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT procedure and for more information about the overloaded DBMS_RULE.EVALUATE procedure

Rule Action Context

An **action context** contains optional information associated with a rule that is interpreted by the client of the **rules engine** when the rule is evaluated for an event. The client of the rules engine can be a user-created application or an internal feature of Oracle, such as Streams. Each rule has only one action context. The information in an action context is of type SYS.RE$NV_LIST, which is a type that contains an array of name-value pairs.

The rule action context information provides a context for the action taken by a client of the rules engine when a rule evaluates to TRUE or MAYBE. The rules engine does not interpret the action context. Instead, it returns the action context, and a client of the rules engine can interpret the action context information.

For example, suppose an event is defined as the addition of a new employee to a company. If the employee information is stored in the hr.employees table, then the event occurs whenever a row is inserted into this table. The company wants to specify that a number of actions are taken when a new employee is added, but the actions depend on which department the employee joins. One of these actions is that the employee is registered for a course relating to the department.

In this scenario, the company can create a rule for each department with an appropriate action context. Here, an action context returned when a rule evaluates to TRUE specifies the number of a course that an employee should take. Here are parts of the **rule condition**s and the action contexts for three departments:

Rule Name	Part of the Rule Condition	Action Context Name-Value Pair
rule_dep_10	department_id = 10	course_number, 1057
rule_dep_20	department_id = 20	course_number, 1215
rule_dep_30	department_id = 30	NULL

These action contexts return the following instructions to the client application:

- The action context for the rule_dep_10 rule instructs the client application to enroll the new employee in course number 1057.
- The action context for the rule_dep_20 rule instructs the client application to enroll the new employee in course number 1215.
- The NULL action context for the rule_dep_30 rule instructs the client application not to enroll the new employee in any course.

Each action context can contain zero or more name-value pairs. If an action context contains more than one name-value pair, then each name in the list must be unique. In this example, the client application to which the rules engine returns the action context registers the new employee in the course with the returned course number. The client application does not register the employee for a course if a NULL action context is returned or if the action context does not contain a course number.

If multiple clients use the same rule, or if you want an action context to return more than one name-value pair, then you can list more than one name-value pair in an action context. For example, suppose the company also adds a new employee to a department electronic mailing list. In this case, the action context for the rule_dep_10 rule might contain two name-value pairs:

Name	Value
course_number	1057
dist_list	admin_list

The following are considerations for names in name-value pairs:

- If different applications use the same action context, then use different names or prefixes of names to avoid naming conflicts.
- Do not use $ and # in names because they can cause conflicts with Oracle-supplied action context names.

You add a name-value pair to an action context using the ADD_PAIR member procedure of the RE$NV_LIST type. You remove a name-value pair from an action context using the REMOVE_PAIR member procedure of the RE$NV_LIST type. If you want to modify an existing name-value pair in an action context, then you should first remove it using the REMOVE_PAIR member procedure and then add an appropriate name-value pair using the ADD_PAIR member procedure.

An action context cannot contain information of the following datatypes:

- CLOB
- NCLOB
- BLOB

- LONG
- LONG RAW

In addition, an action context cannot contain object types with attributes of these datatypes, or object types that use type evolution or type inheritance.

Note: Streams uses action contexts for **custom rule-based transformation**s and, when **subset rule**s are specified, for internal transformations that might be required on LCRs containing UPDATE operations. Streams also uses action contexts to specify a **destination queue** into which an **apply process** enqueues **message**s that satisfy the rule. In addition, Streams uses action contexts to specify whether or not a message that satisfies an apply process rule is executed by the apply process.

See Also:

- "Streams and Action Contexts" on page 6-37
- "Creating a Rule with an Action Context" on page 14-6 and "Altering a Rule" on page 14-6 for examples that add and modify name-value pairs
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the RE$NV_LIST type

Rule Set Evaluation

The **rules engine** evaluates **rule set**s against an event. An event is an occurrence that is defined by the client of the rules engine. The client initiates evaluation of an event by calling the DBMS_RULE.EVALUATE procedure. This procedure enables the client to send some information about the event to the rules engine for evaluation against a rule set. The event itself can have more information than the information that the client sends to the rules engine.

The following information is specified by the client when it calls the DBMS_RULE.EVALUATE procedure:

- The name of the rule set that contains the rules to use to evaluate the event.
- The **evaluation context** to use for evaluation. Only rules that use the specified evaluation context are evaluated.
- Table values and variable values. The table values contain rowids that refer to the data in table rows, and the variable values contain the data for explicit variables. Values specified for implicit variables override the values that might be obtained using a variable value function. If a specified variable has attributes, then the client can send a value for the entire variable, or the client can send values for any number of the attributes of the variable. However, clients cannot specify attribute values if the value of the entire variable is specified.
- An optional **event context**. An event context is a varray of type SYS.RE$NV_LIST that contains name-value pairs that contain information about the event. This optional information is not used directly or interpreted by the rules engine. Instead, it is passed to client callbacks, such as an evaluation function, a variable value function (for implicit variables), and a variable method function.

The client can also send other information about how to evaluate an event against the rule set using the DBMS_RULE.EVALUATE procedure. For example, the caller can specify if evaluation must stop as soon as the first TRUE rule or the first MAYBE rule (if there are no TRUE rules) is found.

If the client wants all of the rules that evaluate to TRUE or MAYBE returned to it, then the client can specify whether evaluation results should be sent back in a complete list of the rules that evaluated to TRUE or MAYBE, or evaluation results should be sent back iteratively. When evaluation results are sent iteratively to the client, the client can retrieve each rule that evaluated to TRUE or MAYBE one by one using the GET_NEXT_HIT function in the DBMS_RULE package.

The rules engine uses the rules in the specified rule set for evaluation and returns the results to the client. The rules engine returns rules using two OUT parameters in the EVALUATE procedure. This procedure is overloaded and the two OUT parameters are different in each version of the procedure:

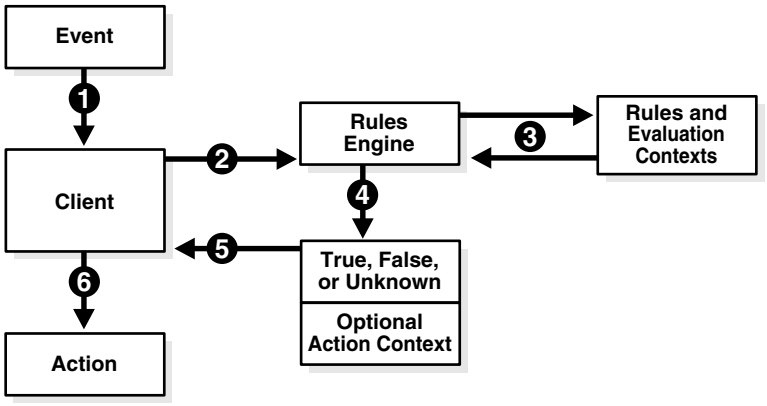
- One version of the procedure returns all of the rules that evaluate to TRUE in one list or all of the rules that evaluate to MAYBE in one list, and the two OUT parameters for this version of the procedure are true_rules and maybe_rules. That is, the true_rules parameter returns rules in one list that evaluate to TRUE, and the maybe_rules parameter returns rules in one list that might evaluate to TRUE given more information.
- The other version of the procedure returns all of the rules that evaluate to TRUE or MAYBE iteratively at the request of the client, and the two OUT parameters for this version of the procedure are true_rules_iterator and maybe_rules_iterator. That is, the true_rules_iterator parameter returns rules that evaluate to TRUE one by one, and the maybe_rules_iterator parameter returns rules one by one that might evaluate to TRUE given more information.

Rule Set Evaluation Process

Figure 5–1 shows the **rule set** evaluation process:

1. A client-defined event occurs.
2. The client initiates evaluation of a rule set by sending information about an event to the **rules engine** using the DBMS_RULE.EVALUATE procedure.
3. The rules engine evaluates the rule set for the event using the relevant **evaluation context**. The client specifies both the rule set and the evaluation context in the call to the DBMS_RULE.EVALUATE procedure. Only rules that are in the specified rule set, and use the specified evaluation context, are used for evaluation.
4. The rules engine obtains the results of the evaluation. Each rule evaluates to either TRUE, FALSE, or NULL (unknown).
5. The rules engine returns rules that evaluated to TRUE to the client, either in a complete list or one by one. Each returned rule is returned with its entire **action context**, which can contain information or can be NULL.
6. The client performs actions based on the results returned by the rules engine. The rules engine does not perform actions based on rule evaluations.

Figure 5–1 Rule Set Evaluation



See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_RULE.EVALUATE` procedure
- "Rule Conditions with Undefined Variables that Evaluate to NULL" on page 6-45 for information about **Streams client**s and `maybe_rules`

Partial Evaluation

Partial evaluation occurs when the `DBMS_RULE.EVALUATE` procedure is run without data for all the tables and variables in the specified **evaluation context**. During partial evaluation, some rules can reference columns, variables, or attributes that are unavailable, while some other rules can reference only available data.

For example, consider a scenario where only the following data is available during evaluation:

- Column `tab1.col = 7`
- Attribute `v1.a1 = 'ABC'`

The following rules are used for evaluation:

- Rule `R1` has the following condition:

```
(tab1.col = 5)
```

- Rule `R2` has the following condition:

```
(:v1.a2 > 'aaa')
```

- Rule `R3` has the following condition:

```
(:v1.a1 = 'ABC') OR (:v2 = 5)
```

- Rule `R4` has the following condition:

```
(:v1.a1 = UPPER('abc'))
```

Given this scenario, `R1` and `R4` reference available data, `R2` references unavailable data, and `R3` references available data and unavailable data.

Partial evaluation always evaluates only simple conditions within a rule. If the **rule condition** has parts which are not simple, then the rule might or might not be evaluated completely, depending on the extent to which data is available. If a rule is not completely evaluated, then it can be returned as a `MAYBE` rule.

Given the rules in this scenario, `R1` and the first part of `R3` are evaluated, but `R2` and `R4` are not evaluated. The following results are returned to the client:

- `R1` evaluates to `FALSE`, and so is not returned.
- `R2` is returned as `MAYBE` because information about attribute `v1.a2` is not available.
- `R3` is returned as `TRUE` because `R3` is a simple rule and the value of `v1.a1` matches the first part of the rule condition.
- `R4` is returned as `MAYBE` because the rule condition is not simple. The client must supply the value of variable `v1` for this rule to evaluate to `TRUE` or `FALSE`.

See Also: "Simple Rule Conditions" on page 5-3

Database Objects and Privileges Related to Rules

You can create the following types of database objects directly using the `DBMS_RULE_ADM` package:

- Evaluation contexts
- Rules
- Rule sets

You can create rules and **rule set**s indirectly using the `DBMS_STREAMS_ADM` package. You control the privileges for these database objects using the following procedures in the `DBMS_RULE_ADM` package:

- `GRANT_OBJECT_PRIVILEGE`
- `GRANT_SYSTEM_PRIVILEGE`
- `REVOKE_OBJECT_PRIVILEGE`
- `REVOKE_SYSTEM_PRIVILEGE`

To allow a user to create rule sets, rules, and **evaluation context**s in the user's own schema, grant the user the following system privileges:

- `CREATE_RULE_SET_OBJ`
- `CREATE_RULE_OBJ`
- `CREATE_EVALUATION_CONTEXT_OBJ`

These privileges, and the privileges discussed in the following sections, can be granted to the user directly or through a role.

Note: When you grant a privilege on "ANY" object (for example, ALTER_ANY_RULE), and the initialization parameter O7_DICTIONARY_ACCESSIBILITY is set to false, you give the user access to that type of object in all schemas except the SYS schema. By default, the initialization parameter O7_DICTIONARY_ACCESSIBILITY is set to false.

If you want to grant access to an object in the SYS schema, then you can grant object privileges explicitly on the object. Alternatively, you can set the O7_DICTIONARY_ACCESSIBILITY initialization parameter to true. Then privileges granted on "ANY" object will allow access to any schema, including SYS.

See Also:

- "The Components of a Rule" on page 5-1 for more information about these database objects
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the system and object privileges for these database objects
- *Oracle Database Concepts* and *Oracle Database Security Guide* for general information about user privileges
- Chapter 6, "How Rules Are Used in Streams" for more information about creating rules and rule sets indirectly using the DBMS_STREAMS_ADM package

Privileges for Creating Database Objects Related to Rules

To create an **evaluation context**, rule, or **rule set** in a schema, a user must meet at least one of the following conditions:

- The schema must be the user's own schema, and the user must be granted the create system privilege for the type of database object being created. For example, to create a rule set in the user's own schema, a user must be granted the CREATE_RULE_SET_OBJ system privilege.
- The user must be granted the create any system privilege for the type of database object being created. For example, to create an evaluation context in any schema, a user must be granted the CREATE_ANY_EVALUATION_CONTEXT system privilege.

Note: When creating a rule with an evaluation context, the rule owner must have privileges on all objects accessed by the evaluation context.

Privileges for Altering Database Objects Related to Rules

To alter an **evaluation context**, rule, or **rule set**, a user must meet at least one of the following conditions:

- The user must own the database object.
- The user must be granted the alter object privilege for the database object if it is in another user's schema. For example, to alter a rule set in another user's schema, a user must be granted the `ALTER_ON_RULE_SET` object privilege on the rule set.
- The user must be granted the alter any system privilege for the database object. For example, to alter a rule in any schema, a user must be granted the `ALTER_ANY_RULE` system privilege.

Privileges for Dropping Database Objects Related to Rules

To drop an **evaluation context**, rule, or **rule set**, a user must meet at least one of the following conditions:

- The user must own the database object.
- The user must be granted the drop any system privilege for the database object. For example, to drop a rule set in any schema, a user must be granted the `DROP_ANY_RULE_SET` system privilege.

Privileges for Placing Rules in a Rule Set

This section describes the privileges required to place a rule in a **rule set**. The user must meet at least one of the following conditions for the rule:

- The user must own the rule.
- The user must be granted the execute object privilege on the rule if the rule is in another user's schema. For example, to place a rule named `depts` in the `hr` schema in a rule set, a user must be granted the `EXECUTE_ON_RULE` privilege for the `hr.depts` rule.
- The user must be granted the execute any system privilege for rules. For example, to place any rule in a rule set, a user must be granted the `EXECUTE_ANY_RULE` system privilege.

The user also must meet at least one of the following conditions for the rule set:

- The user must own the rule set.
- The user must be granted the alter object privilege on the rule set if the rule set is in another user's schema. For example, to place a rule in the `human_resources` rule set in the `hr` schema, a user must be granted the `ALTER_ON_RULE_SET` privilege for the `hr.human_resources` rule set.
- The user must be granted the alter any system privilege for rule sets. For example, to place a rule in any rule set, a user must be granted the `ALTER_ANY_RULE_SET` system privilege.

In addition, the rule owner must have privileges on all objects referenced by the rule. These privileges are important when the rule does not have an **evaluation context** associated with it.

Privileges for Evaluating a Rule Set

To evaluate a **rule set**, a user must meet at least one of the following conditions:

- The user must own the rule set.
- The user must be granted the execute object privilege on the rule set if it is in another user's schema. For example, to evaluate a rule set named `human_resources` in the `hr` schema, a user must be granted the `EXECUTE_ON_RULE_SET` privilege for the `hr.human_resources` rule set.
- The user must be granted the execute any system privilege for rule sets. For example, to evaluate any rule set, a user must be granted the `EXECUTE_ANY_RULE_SET` system privilege.

Granting `EXECUTE` object privilege on a rule set requires that the grantor have the `EXECUTE` privilege specified `WITH GRANT OPTION` on all rules currently in the rule set.

Privileges for Using an Evaluation Context

To use an **evaluation context** in a rule or a **rule set**, the user who owns the rule or rule set must meet at least one of the following conditions for the evaluation context:

- The user must own the evaluation context.
- The user must be granted the `EXECUTE_ON_EVALUATION_CONTEXT` privilege on the evaluation context, if it is in another user's schema.
- The user must be granted the `EXECUTE_ANY_EVALUATION_CONTEXT` system privilege for evaluation contexts.

6

How Rules Are Used in Streams

This chapter explains how **rule**s are used in Streams.

This chapter contains these topics:

See Also:

- Chapter 5, "Rules" for more information about rules
- Chapter 14, "Managing Rules"

Overview of How Rules Are Used in Streams

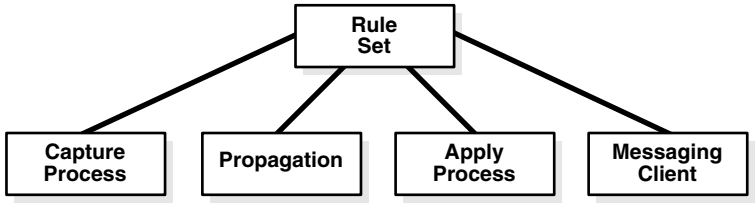
In Streams, each of the following mechanisms is called a **Streams client** because each one is a client of a **rules engine**, when the mechanism is associated with one or more **rule set**s:

- Capture process
- Propagation
- Apply process
- Messaging client

Each of these clients can be associated with at most two rule sets: a **positive rule set** and a **negative rule set**. A single rule set can be used by multiple **capture process**es, **propagation**s, **apply process**es, and **messaging client**s within the same database. Also, a single rule set can be a positive rule set for one Streams client and a negative rule set for another Streams client.

Figure 6–1 illustrates how multiple clients of a rules engine can use one rule set.

Figure 6–1 One Rule Set Can Be Used by Multiple Clients of a Rules Engine



A Streams client performs a task if a **message** satisfies its rule sets. In general, a message satisfies the rule sets for a Streams client if *no rules* in the negative rule set evaluate to `TRUE` for the message, and *at least one rule* in the positive rule set evaluates to `TRUE` for the message.

"Rule Sets and Rule Evaluation of Messages" on page 6-3 contains more detailed information about how a message satisfies the rule sets for a Streams client, including information about Streams client behavior when one or more rule sets are not specified.

Specifically, you use rule sets in Streams to do the following:

- Specify the changes that a capture process captures from the redo log or discards. That is, if a change found in the redo log satisfies the rule sets for a capture process, then the capture process captures the change. If a change found in the redo log causes does not satisfy the rule sets for a capture process, then the capture process discards the change.
- Specify the messages that a propagation propagates from one **queue** to another or discards. That is, if a message in a queue satisfies the rule sets for a propagation, then the propagation propagates the message. If a message in a queue does not satisfy the rule sets for a propagation, then the propagation discards the message.
- Specify the messages that an apply process retrieves from a queue or discards. That is, if a message in a queue satisfies the rule sets for an apply process, then the message is dequeued and processed by the apply process. If a message in a queue does not satisfy the rule sets for an apply process, then the apply process discards the message.
- Specify the **user-enqueued message**s that a **messaging client** dequeues from a queue or discards. That is, if a user-enqueued message in a queue satisfies the rule sets for a messaging client, then the user or application that is using the messaging client dequeues the message. If a user-enqueued message in a queue does not satisfy the rule sets for a messaging client, then the user or application that is using the messaging client discards the message.

In the case of a propagation or an apply process, the messages evaluated against the rule sets can be **captured message**s or user-enqueued messages.

If there are conflicting **rule**s in the positive rule set associated with a client, then the client performs the task if either rule evaluates to `TRUE`. For example, if a rule in the positive rule set for a capture process contains one rule that instructs the capture process to capture the results of data manipulation language (DML) changes to the `hr.employees` table, but another rule in the rule set instructs the capture process not to capture the results of DML changes to the `hr.employees` table, then the capture process captures these changes.

Similarly, if there are conflicting rules in the negative rule set associated with a client, then the client discards a message if either rule evaluates to `TRUE` for the message. For example, if a rule in the negative rule set for a capture process contains one rule that instructs the capture process to discard the results of DML changes to the `hr.departments` table, but another rule in the rule set instructs the capture process not to discard the results of DML changes to the `hr.departments` table, then the capture process discards these changes.

See Also: For more information about Streams clients:

- Chapter 2, "Streams Capture Process"
- "Message Propagation Between Queues" on page 3-4
- Chapter 4, "Streams Apply Process"
- "Messaging Clients" on page 3-10

Rule Sets and Rule Evaluation of Messages

Streams clients perform the following tasks based on **rule**s:

- A **capture process** captures changes in the redo log, converts the changes into logical change records (LCRs), and enqueues **message**s containing these LCRs into the capture process **queue**.
- A **propagation** propagates either **captured message**s or **user-enqueued message**s, or both, from a **source queue** to a **destination queue**.
- An **apply process** dequeues either captured or user-enqueued messages from its queue and applies these messages directly or sends the messages to an **apply handler**.
- A **messaging client** dequeues user-enqueued messages from its queue.

These Streams clients are all clients of the **rules engine**. A Streams client performs its task for a message when the message satisfies the **rule set**s used by the Streams client. A Streams client can have no rule set, only a **positive rule set**, only a **negative rule set**, or both a positive and a negative rule set. The following sections explain how rule evaluation works in each of these cases:

- Streams Client with No Rule Set
- Streams Client with a Positive Rule Set Only
- Streams Client with a Negative Rule Set Only
- Streams Client with Both a Positive and a Negative Rule Set
- Streams Client with One or More Empty Rule Sets
- Summary of Rule Sets and Streams Client Behavior

Streams Client with No Rule Set

A **Streams client** with no **rule set** performs its task for all of the **message**s it encounters. An empty rule set is not the same as no rule set at all.

> **See Also:** "Streams Client with One or More Empty Rule Sets" on page 6-4

Streams Client with a Positive Rule Set Only

A **Streams client** with a **positive rule set**, but no **negative rule set**, performs its task for a **message** if any **rule** in the positive rule set evaluates to `TRUE` for the message. However, if all of the rules in a positive rule set evaluate to `FALSE` for the message, then the Streams client discards the message.

Streams Client with a Negative Rule Set Only

A **Streams client** with a **negative rule set**, but no **positive rule set**, discards a **message** if any **rule** in the negative rule set evaluates to `TRUE` for the message. However, if all of the rules in a negative rule set evaluate to `FALSE` for the message, then the Streams client performs its task for the message.

Streams Client with Both a Positive and a Negative Rule Set

If **Streams client** has both a positive and a **negative rule set**, then the negative rule set is evaluated first for a **message**. If any **rule** in the negative rule set evaluates to `TRUE` for the message, then the message is discarded, and the message is never evaluated against the **positive rule set**.

However, if all of the rules in the negative rule set evaluate to `FALSE` for the message, then the message is evaluated against the positive rule set. At this point, the behavior is the same as when the Streams client only has a positive rule set. That is, the Streams client performs its task for a message if any rule in the positive rule set evaluates to `TRUE` for the message. If all of the rules in a positive rule set evaluate to `FALSE` for the message, then the Streams client discards the message.

Streams Client with One or More Empty Rule Sets

A **Streams client** can have one or more empty **rule set**s. A Streams client behaves in the following ways if it has one or more empty rule sets:

- If a Streams client has no **positive rule set**, and its **negative rule set** is empty, then the Streams client performs its task for all **message**s.
- If a Streams client has both a positive and a negative rule set, and the negative rule set is empty but its positive rule set contains **rule**s, then the Streams client performs its task based on the rules in the positive rule set.
- If a Streams client has a positive rule set that is empty, then the Streams client discards all messages, regardless of the state of its negative rule set.

Summary of Rule Sets and Streams Client Behavior

Table 6–1 summarizes the **Streams client** behavior described in the previous sections.

Table 6–1 Rule Sets and Streams Client Behavior

Negative Rule Set	Positive Rule Set	Streams Client Behavior
None	None	Performs its task for all **message**s
None	Exists with rules	Performs its task for messages that evaluate to `TRUE` against the **positive rule set**
Exists with rules	None	Discards messages that evaluate to `TRUE` against the **negative rule set**, and performs its task for all other messages
Exists with rules	Exists with rules	Discards messages that evaluate to `TRUE` against the negative rule set, and performs its task for remaining messages that evaluate to `TRUE` against the positive rule set. The negative rule set is evaluated first.
Exists but is empty	None	Performs its task for all messages
Exists but is empty	Exists with rules	Performs its task for messages that evaluate to `TRUE` against the positive rule set
None	Exists but is empty	Discards all messages
Exists but is empty	Exists but is empty	Discards all messages
Exists with rules	Exists but is empty	Discards all messages

System-Created Rules

A **Streams client** performs its task for a **message** if the message satisfies its **rule set**s. A **system-created rule** is created by the `DBMS_STREAMS_ADM` package and can specify one of the following levels of granularity: table, schema, or global. This section describes each of these levels. You can specify more than one level for a particular task. For example, you can instruct a single **apply process** to perform table-level apply for specific tables in the `oe` schema and schema-level apply for the entire `hr` schema. In addition, a single **rule** pertains to either the results of data manipulation language (DML) changes or data definition language (DDL) changes. So, for example, you must use at least two system-created rules to include all of the changes to a particular table: one rule for the results of DML changes and another rule for DDL changes. The results of a DML change are the row changes recorded in the redo log because of the DML change, or the row LCRs in a **queue** that encapsulate each row change.

Table 6–2 shows what each level of rule means for each Streams task. Remember that a **negative rule set** is evaluated before a **positive rule set**.

Table 6–2 Types of Tasks and Rule Levels

Task	Table Rule	Schema Rule	Global Rule
Capture with a **capture process**	If the **table rule** is in a negative rule set, then discard the changes in the redo log for the specified table. If the table rule is in a positive rule set, then capture all or a subset of the changes in the redo log for the specified table, convert them into logical change records (LCRs), and enqueue them.	If the **schema rule** is in a negative rule set, then discard the changes in the redo log for the schema itself and for the database objects in the specified schema. If the schema rule is in a positive rule set, then capture the changes in the redo log for the schema itself and for the database objects in the specified schema, convert them into LCRs, and enqueue them.	If the **global rule** is in a negative rule set, then discard the changes to all of the database objects in the database. If the global rule is in a positive rule set, then capture the changes to all of the database objects in the database, convert them into LCRs, and enqueue them.
Propagate with a **propagation**	If the table rule is in a negative rule set, then discard the LCRs relating to the specified table in the **source queue**. If the table rule is in a positive rule set, then propagate all or a subset of the LCRs relating to the specified table in the source queue to the **destination queue**.	If the schema rule is in a negative rule set, then discard the LCRs related to the specified schema itself and the LCRs related to database objects in the schema in the source queue. If the schema rule is in a positive rule set, then propagate the LCRs related to the specified schema itself and the LCRs related to database objects in the schema in the source queue to the destination queue.	If the global rule is in a negative rule set, then discard all of the LCRs in the source queue. If the global rule is in a positive rule set, then propagate all of the LCRs in the source queue to the destination queue.
Apply with an **apply process**	If the table rule is in a negative rule set, then discard the LCRs in the queue relating to the specified table. If the table rule is in a positive rule set, then apply all or a subset of the LCRs in the queue relating to the specified table.	If the schema rule is in a negative rule set, then discard the LCRs in the queue relating to the specified schema itself and the database objects in the schema. If the schema rule is in a positive rule set, then apply the LCRs in the queue relating to the specified schema itself and the database objects in the schema.	If the global rule is in a negative rule set, then discard all of the LCRs in the queue. If the global rule is in a positive rule set, then apply all of the LCRs in the queue.
Dequeue with a **messaging client**	If the table rule is in a negative rule set, then, when the messaging client is invoked, discard the user-enqueued LCRs relating to the specified table in the queue. If the table rule is in a positive rule set, then, when the messaging client is invoked, dequeue all or a subset of the user-enqueued LCRs relating to the specified table in the queue.	If the schema rule is in a negative rule set, then, when the messaging client is invoked, discard the user-enqueued LCRs relating to the specified schema itself and the database objects in the schema in the queue. If the schema rule is in a positive rule set, then, when the messaging client is invoked, dequeue the user-enqueued LCRs relating to the specified schema itself and the database objects in the schema in the queue.	If the global rule is in a negative rule set, then, when the messaging client is invoked, discard all of the user-enqueued LCRs in the queue. If the global rule is in a positive rule set, then, when the messaging client is invoked, dequeue all of the user-enqueued LCRs in the queue.

You can use procedures in the `DBMS_STREAMS_ADM` package to create rules at each of these levels. A system-created rule can include conditions that modify the Streams client behavior beyond the descriptions in Table 6–2. For example, some rules can specify a particular **source database** for LCRs, and, in this case, the rule evaluates to

TRUE only if an LCR originated at the specified source database. Table 6–3 lists the types of system-created **rule condition**s that can be specified in the rules created by the DBMS_STREAMS_ADM package.

Table 6–3 System-Created Rule Conditions Created by DBMS_STREAMS_ADM Package

Rule Condition Evaluates to TRUE for	Streams Client	Create Using Procedure
All row changes recorded in the redo log because of DML changes to any of the tables in a particular database	Capture Process	ADD_GLOBAL_RULES
All DDL changes recorded in the redo log to any of the database objects in a particular database	Capture Process	ADD_GLOBAL_RULES
All row changes recorded in the redo log because of DML changes to any of the tables in a particular schema	Capture Process	ADD_SCHEMA_RULES
All DDL changes recorded in the redo log to a particular schema and any of the database objects in the schema	Capture Process	ADD_SCHEMA_RULES
All row changes recorded in the redo log because of DML changes to a particular table	Capture Process	ADD_TABLE_RULES
All DDL changes recorded in the redo log to a particular table	Capture Process	ADD_TABLE_RULES
All row changes recorded in the redo log because of DML changes to a subset of rows in a particular table	Capture Process	ADD_SUBSET_RULES
All row LCRs in the source queue	Propagation	ADD_GLOBAL_PROPAGATION_RULES
All DDL LCRs in the source queue	Propagation	ADD_GLOBAL_PROPAGATION_RULES
All row LCRs in the source queue relating to the tables in a particular schema	Propagation	ADD_SCHEMA_PROPAGATION_RULES
All DDL LCRs in the source queue relating to a particular schema and any of the database objects in the schema	Propagation	ADD_SCHEMA_PROPAGATION_RULES
All row LCRs in the source queue relating to a particular table	Propagation	ADD_TABLE_PROPAGATION_RULES
All DDL LCRs in the source queue relating to a particular table	Propagation	ADD_TABLE_PROPAGATION_RULES
All row LCRs in the source queue relating to a subset of rows in a particular table	Propagation	ADD_SUBSET_PROPAGATION_RULES
All **user-enqueued message**s in the source queue of the specified type that satisfy the user-specified rule condition	Propagation	ADD_MESSAGE_PROPAGATION_RULE
All row LCRs in the queue used by the apply process	Apply Process	ADD_GLOBAL_RULES
All DDL LCRs in the queue used by the apply process	Apply Process	ADD_GLOBAL_RULES
All row LCRs in the queue used by the apply process relating to the tables in a particular schema	Apply Process	ADD_SCHEMA_RULES

Table 6–3 (Cont.) System-Created Rule Conditions Created by DBMS_STREAMS_ADM Package

Rule Condition Evaluates to TRUE for	Streams Client	Create Using Procedure
All DDL LCRs in the queue used by the apply process relating to a particular schema and any of the database objects in the schema	Apply Process	ADD_SCHEMA_RULES
All row LCRs in the queue used by the apply process relating to a particular table	Apply Process	ADD_TABLE_RULES
All DDL LCRs in the queue used by the apply process relating to a particular table	Apply Process	ADD_TABLE_RULES
All row LCRs in the queue used by the apply process relating to a subset of rows in a particular table	Apply Process	ADD_SUBSET_RULES
All user-enqueued messages in the queue used by the apply process of the specified type that satisfy the user-specified rule condition	Apply Process	ADD_MESSAGE_RULE
All user-enqueued row LCRs in the queue used by the messaging client	Messaging Client	ADD_GLOBAL_RULES
All user-enqueued DDL LCRs in the queue used by the messaging client	Messaging Client	ADD_GLOBAL_RULES
All user-enqueued row LCRs in the queue used by the messaging client relating to the tables in a particular schema	Messaging Client	ADD_SCHEMA_RULES
All user-enqueued DDL LCRs in the queue used by the messaging client relating to a particular schema and any of the database objects in the schema	Messaging Client	ADD_SCHEMA_RULES
All user-enqueued row LCRs in the messaging client's queue relating to a particular table	Messaging Client	ADD_TABLE_RULES
All user-enqueued DDL LCRs in the queue used by the messaging client relating to a particular table	Messaging Client	ADD_TABLE_RULES
All user-enqueued row LCRs in the queue used by the messaging client relating to a subset of rows in a particular table	Messaging Client	ADD_SUBSET_RULES
All user-enqueued messages in the queue used by the messaging client of the specified type that satisfy the user-specified rule condition	Messaging Client	ADD_MESSAGE_RULE

Each procedure listed in Table 6–3 does the following:

- Creates a **capture process**, **propagation**, **apply process**, or **messaging client** if it does not already exist.
- Creates a rule set for the specified capture process, propagation, apply process, or messaging client if a rule set does not already exist for it. The rule set can be a positive rule set or a negative rule set. You can create each type of rule set by running the procedure at least twice.
- Creates zero or more rules and adds the rules to the rule set for the specified capture process, propagation, apply process, or messaging client. Based on your specifications when you run one of these procedures, the procedure adds the rules either to the positive rule set or to the negative rule set.

Except for the ADD_MESSAGE_RULE and ADD_MESSAGE_PROPAGATION_RULE procedures, these procedures create rule sets that use the SYS.STREAMS$_EVALUATION_CONTEXT **evaluation context**, which is an Oracle-supplied evaluation context for Streams environments. Global, schema, table, and **subset rule**s use the SYS.STREAMS$_EVALUATION_CONTEXT evaluation context.

However, when you create a rule using either the ADD_MESSAGE_RULE or the ADD_MESSAGE_PROPAGATION_RULE procedure, the rule uses a system-generated evaluation context that is customized specifically for each message type. Rule sets created by the ADD_MESSAGE_RULE or the ADD_MESSAGE_PROPAGATION_RULE procedure do not have an evaluation context.

Except for ADD_SUBSET_RULES, ADD_SUBSET_PROPAGATION_RULES, ADD_MESSAGE_RULE, and ADD_MESSAGE_PROPAGATION_RULE, these procedures create either zero, one, or two rules. If you want to perform the Streams task for only the row changes resulting from DML changes or only for only DDL changes, then only one rule is created. If, however, you want to perform the Streams task for both the results of DML changes and DDL changes, then a rule is created for each. If you create a DML rule for a table now, then you can create a DDL rule for the same table in the future without modifying the DML rule created earlier. The same applies if you create a DDL rule for a table first and a DML rule for the same table in the future.

The ADD_SUBSET_RULES and ADD_SUBSET_PROPAGATION_RULES procedures always create three rules for three different types of DML operations on a table: INSERT, UPDATE, and DELETE. These procedures do not create rules for DDL changes to a table. You can use the ADD_TABLE_RULES or ADD_TABLE_PROPAGATION_RULES procedure to create a DDL rule for a table. In addition, you can add subset rules to positive rule sets only, not to negative rule sets.

The ADD_MESSAGE_RULE and ADD_MESSAGE_PROPAGATION_RULE procedures always create one rule with a user-specified rule condition. These procedures create rules for user-enqueued messages. They do not create rules for the results of DML changes or DDL changes to a table.

When you create propagation rules for **captured message**s, Oracle recommends that you specify a source database for the changes. An apply process uses transaction control messages to assemble captured messages into committed transactions. These transaction control messages, such as COMMIT and ROLLBACK, contain the name of the source database where the message occurred. To avoid unintended cycling of these messages, propagation rules should contain a condition specifying the source database, and you accomplish this by specifying the source database when you create the propagation rules.

The following sections describe system-created rules in more detail:

- Global Rules
- Schema Rules
- Table Rules
- Subset Rules
- Message Rules
- System-Created Rules and Negative Rule Sets
- System-Created Rules with Added User-Defined Conditions

Note:

- To create rules with more complex rule conditions, such as rules that use the `NOT` or `OR` logical conditions, either use the `and_condition` parameter, which is available with some of the procedures in the `DBMS_STREAMS_ADM` package, or use the `DBMS_RULE_ADM` package.
- Each example in the sections that follow should be completed by a Streams administrator that has been granted the appropriate privileges, unless specified otherwise.
- Some of the examples in this section have additional prerequisites. For example, a queue specified by a procedure parameter must exist.

See Also:

- "Rule Sets and Rule Evaluation of Messages" on page 6-3 for information about how messages satisfy the rule sets for a Streams client
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_STREAMS_ADM` package and the `DBMS_RULE_ADM` package
- "Evaluation Contexts Used in Streams" on page 6-34
- "Logical Change Records (LCRs)" on page 2-2
- "Complex Rule Conditions" on page 6-43

Global Rules

When you use a **rule** to specify a Streams task that is relevant either to an entire database or to an entire **queue**, you are specifying a **global rule**. You can specify a global rule for DML changes, a global rule for DDL changes, or a global rule for each type of change (two rules total).

A single global rule in the **positive rule set** for a **capture process** means that the capture process captures the results of either all DML changes or all DDL changes to the **source database**. A single global rule in the **negative rule set** for a capture process means that the capture process discards the results of either all DML changes or all DDL changes to the source database.

A single global rule in the positive rule set for a **propagation** means that the propagation propagates either all row LCRs or all DDL LCRs in the **source queue** to the **destination queue**. A single global rule in the negative rule set for a propagation means that the propagation discards either all row LCRs or all DDL LCRs in the source queue.

A single global rule in the positive rule set for an **apply process** means that the apply process applies either all row LCRs or all DDL LCRs in its queue for a specified source database. A single global rule in the negative rule set for an apply process means that the apply process discards either all row LCRs or all DDL LCRs in its queue for a specified source database.

If you want to use global rules, but you are concerned about changes to database objects that are not supported by Streams, then you can create rules using the `DBMS_RULE_ADM` package to discard unsupported changes.

> **See Also:** "Rule Conditions that Instruct Streams Clients to Discard Unsupported LCRs" on page 6-42

Global Rules Example

Suppose you use the `ADD_GLOBAL_RULES` procedure in the `DBMS_STREAMS_ADM` package to instruct a Streams capture process to capture all DML changes and DDL changes in a database.

Run the `ADD_GLOBAL_RULES` procedure to create the rules:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_GLOBAL_RULES(
    streams_type        =>  'capture',
    streams_name        =>  'capture',
    queue_name          =>  'streams_queue',
    include_dml         =>  true,
    include_ddl         =>  true,
    include_tagged_lcr  =>  false,
    source_database     =>  NULL,
    inclusion_rule      =>  true);
END;
/
```

Notice that the `inclusion_rule` parameter is set to `true`. This setting means that the **system-created rule**s are added to the positive rule set for the capture process.

`NULL` can be specified for the `source_database` parameter because rules are being created for a **local capture process**. You can also specify the global name of the local database. When creating rules for a **downstream capture process** or apply process using `ADD_GLOBAL_RULES`, specify a source database name.

The `ADD_GLOBAL_RULES` procedure creates two rules: one for row LCRs (which contain the results of DML changes) and one for DDL LCRs.

Here is the **rule condition** used by the row LCR rule:

```
(:dml.is_null_tag() = 'Y' )
```

Notice that the condition in the DML rule begins with the variable `:dml`. The value is determined by a call to the specified member function for the row LCR being evaluated. So, `:dml.is_null_tag()` is a call to the `IS_NULL_TAG` member function for the row LCR being evaluated.

Here is the rule condition used by the DDL LCR rule:

```
(:ddl.is_null_tag() = 'Y' )
```

Notice that the condition in the DDL rule begins with the variable `:ddl`. The value is determined by a call to the specified member function for the DDL LCR being evaluated. So, `:ddl.is_null_tag()` is a call to the `IS_NULL_TAG` member function for the DDL LCR being evaluated.

For a capture process, these conditions indicate that the **tag** must be `NULL` in a redo record for the capture process to capture a change. For a propagation, these conditions indicate that the tag must be `NULL` in an LCR for the propagation to propagate the LCR. For an apply process, these conditions indicate that the tag must be `NULL` in an LCR for the apply process to apply the LCR.

Given the rules created by this example in the positive rule set for the capture process, the capture process captures all supported DML and DDL changes made to the database.

Caution: If you add global rules to the positive rule set for a capture process, then make sure you add rules to the negative capture process rule set to exclude database objects that are not support by Streams. Query the `DBA_STREAMS_UNSUPPORTED` data dictionary view to determine which database objects are not supported by Streams. If unsupported database objects are not excluded, then capture errors will result.

See Also: "Listing the Database Objects that Are Not Compatible with Streams" on page 26-7

System-Created Global Rules Avoid Empty Rule Conditions Automatically

You can omit the `is_null_tag` condition in system-created rules by specifying `true` for the `include_tagged_lcr` parameter when you run a procedure in the `DBMS_STREAMS_ADM` package. For example, the following `ADD_GLOBAL_RULES` procedure creates rules without the `is_null_tag` condition:

```
BEGIN DBMS_STREAMS_ADM.ADD_GLOBAL_RULES(
   streams_type        =>  'capture',
   streams_name        =>  'capture_002',
   queue_name          =>  'streams_queue',
   include_dml         =>  true,
   include_ddl         =>  true,
   include_tagged_lcr  =>  true,
   source_database     =>  NULL,
   inclusion_rule      =>  true);
END;
/
```

When you set the `include_tagged_lcr` parameter to `true` for a global rule, and the `source_database_name` parameter is set to `NULL`, the rule condition used by the row LCR rule is the following:

```
(( :dml.get_source_database_name()>=' ' OR
:dml.get_source_database_name()<=' ') )
```

Here is the rule condition used by the DDL LCR rule:

```
(( :ddl.get_source_database_name()>=' ' OR
:ddl.get_source_database_name()<=' ') )
```

The system-created global rules contain these conditions to enable all row and DDL LCRs to evaluate to `TRUE`.

These rule conditions are specified to avoid `NULL` rule conditions for these rules. `NULL` rule conditions are not supported. In this case, if you want to capture all DML and DDL changes to a database, and you do not want to use any **rule-based transformation**s for these changes upon capture, then you can choose to run the capture process without a positive rule set instead of specifying global rules.

Note:

- When you create a capture process using a procedure in the `DBMS_STREAMS_ADM` package and generate one or more rules for the capture process, the objects for which changes are captured are prepared for **instantiation** automatically, unless it is a downstream capture process and there is no database link from the **downstream database** to the source database.
- The capture process does not capture some types of DML and DDL changes, and it does not capture changes made in the `SYS`, `SYSTEM`, or `CTXSYS` schemas.

See Also:

- *Oracle Streams Replication Administrator's Guide* for more information about capture process rules and preparation for instantiation
- Chapter 2, "Streams Capture Process" for more information about the capture process and for detailed information about which DML and DDL statements are captured by a capture process
- Chapter 5, "Rules" for more information about variables in conditions
- *Oracle Streams Replication Administrator's Guide* for more information about Streams tags
- "Rule Sets and Rule Evaluation of Messages" on page 6-3 for more information about running a capture process with no positive rule set

Schema Rules

When you use a **rule** to specify a Streams task that is relevant to a schema, you are specifying a **schema rule**. You can specify a schema rule for DML changes, a schema rule for DDL changes, or a schema rule for each type of change to the schema (two rules total).

A single schema rule in the **positive rule set** for a **capture process** means that the capture process captures either the DML changes or the DDL changes to the schema. A single schema rule in the **negative rule set** for a capture process means that the capture process discards either the DML changes or the DDL changes to the schema.

A single schema rule in the positive rule set for a **propagation** means that the propagation propagates either the row LCRs or the DDL LCRs in the **source queue** that contain changes to the schema. A single schema rule in the negative rule set for a propagation means that the propagation discards either the row LCRs or the DDL LCRs in the source queue that contain changes to the schema.

A single schema rule in the positive rule set for an **apply process** means that the apply process applies either the row LCRs or the DDL LCRs in its **queue** that contain changes to the schema. A single schema rule in the negative rule set for an apply process means that the apply process discards either the row LCRs or the DDL LCRs in its queue that contain changes to the schema.

If you want to use schema rules, but you are concerned about changes to database objects in a schema that are not supported by Streams, then you can create rules using the DBMS_RULE_ADM package to discard unsupported changes.

See Also: "Rule Conditions that Instruct Streams Clients to Discard Unsupported LCRs" on page 6-42

Schema Rule Example

Suppose you use the ADD_SCHEMA_PROPAGATION_RULES procedure in the DBMS_STREAMS_ADM package to instruct a Streams propagation to propagate row LCRs and DDL LCRs relating to the hr schema from a queue at the dbs1.net database to a queue at the dbs2.net database.

Run the ADD_SCHEMA_PROPAGATION_RULES procedure at dbs1.net to create the rules:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES(
    schema_name              =>  'hr',
    streams_name             =>  'dbs1_to_dbs2',
    source_queue_name        =>  'streams_queue',
    destination_queue_name   =>  'streams_queue@dbs2.net',
    include_dml              =>  true,
    include_ddl              =>  true,
    include_tagged_lcr       =>  false,
    source_database          =>  'dbs1.net',
    inclusion_rule           =>  true);
END;
/
```

Notice that the inclusion_rule parameter is set to true. This setting means that the **system-created rule**s are added to the positive rule set for the propagation.

The ADD_SCHEMA_PROPAGATION_RULES procedure creates two rules: one for row LCRs (which contain the results of DML changes) and one for DDL LCRs.

Here is the **rule condition** used by the row LCR rule:

```
((:dml.get_object_owner() = 'HR') and :dml.is_null_tag() = 'Y'
and :dml.get_source_database_name() = 'DBS1.NET' )
```

Here is the rule condition used by the DDL LCR rule:

```
((:ddl.get_object_owner() = 'HR' or :ddl.get_base_table_owner() = 'HR')
and :ddl.is_null_tag() = 'Y' and :ddl.get_source_database_name() = 'DBS1.NET' )
```

The GET_BASE_TABLE_OWNER member function is used in the DDL LCR rule because the GET_OBJECT_OWNER function can return NULL if a user who does not own an object performs a DDL change on the object.

Given these rules in the positive rule set for the propagation, the following list provides examples of changes propagated by the propagation:

- A row is inserted into the `hr.countries` table.
- The `hr.loc_city_ix` index is altered.
- The `hr.employees` table is truncated.
- A column is added to the `hr.countries` table.
- The `hr.update_job_history` trigger is altered.
- A new table named `candidates` is created in the `hr` schema.
- Twenty rows are inserted into the `hr.candidates` table.

The propagation propagates the LCRs that contain all of the changes previously listed from the source queue to the **destination queue**.

Now, given the same rules, suppose a row is inserted into the `oe.inventories` table. This change is ignored because the `oe` schema was not specified in a schema rule, and the `oe.inventories` table was not specified in a table rule.

Table Rules

When you use a **rule** to specify a Streams task that is relevant only for an individual table, you are specifying a **table rule**. You can specify a table rule for DML changes, a table rule for DDL changes, or a table rule for each type of change to a specific table (two rules total).

A single table rule in the **positive rule set** for a **capture process** means that the capture process captures the results of either the DML changes or the DDL changes to the table. A single table rule in the **negative rule set** for a capture process means that the capture process discards the results of either the DML changes or the DDL changes to the table.

A single table rule in the positive rule set for a **propagation** means that the propagation propagates either the row LCRs or the DDL LCRs in the **source queue** that contain changes to the table. A single table rule in the negative rule set for a propagation means that the propagation discards either the row LCRs or the DDL LCRs in the source queue that contain changes to the table.

A single table rule in the positive rule set for an **apply process** means that the apply process applies either the row LCRs or the DDL LCRs in its **queue** that contain changes to the table. A single table rule in the negative rule set for an apply process means that the apply process discards either the row LCRs or the DDL LCRs in its queue that contain changes to the table.

Table Rules Example

Suppose you use the `ADD_TABLE_RULES` procedure in the `DBMS_STREAMS_ADM` package to instruct a Streams apply process to behave in the following ways:

- Apply All Row LCRs Related to the hr.locations Table
- Apply All DDL LCRs Related to the hr.countries Table

Apply All Row LCRs Related to the hr.locations Table The changes in these row LCRs originated at the dbs1.net **source database**.

Run the ADD_TABLE_RULES procedure to create this rule:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name          =>  'hr.locations',
    streams_type        =>  'apply',
    streams_name        =>  'apply',
    queue_name          =>  'streams_queue',
    include_dml         =>  true,
    include_ddl         =>  false,
    include_tagged_lcr  =>  false,
    source_database     =>  'dbs1.net',
    inclusion_rule      =>  true);
END;
/
```

Notice that the inclusion_rule parameter is set to true. This setting means that the **system-created rule** is added to the positive rule set for the apply process.

The ADD_TABLE_RULES procedure creates a rule with a **rule condition** similar to the following:

```
(((:dml.get_object_owner() = 'HR' and :dml.get_object_name() = 'LOCATIONS'))
and :dml.is_null_tag() = 'Y' and :dml.get_source_database_name() = 'DBS1.NET' )
```

Apply All DDL LCRs Related to the hr.countries Table The changes in these DDL LCRs originated at the dbs1.net source database.

Run the ADD_TABLE_RULES procedure to create this rule:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name          =>  'hr.countries',
    streams_type        =>  'apply',
    streams_name        =>  'apply',
    queue_name          =>  'streams_queue',
    include_dml         =>  false,
    include_ddl         =>  true,
    include_tagged_lcr  =>  false,
    source_database     =>  'dbs1.net',
    inclusion_rule      =>  true);
END;
/
```

Notice that the inclusion_rule parameter is set to true. This setting means that the system-created rule is added to the positive rule set for the apply process.

The ADD_TABLE_RULES procedure creates a rule with a rule condition similar to the following:

```
(((:ddl.get_object_owner() = 'HR' and :ddl.get_object_name() = 'COUNTRIES')
or (:ddl.get_base_table_owner() = 'HR'
and :ddl.get_base_table_name() = 'COUNTRIES')) and :ddl.is_null_tag() = 'Y'
and :ddl.get_source_database_name() = 'DBS1.NET' )
```

The GET_BASE_TABLE_OWNER and GET_BASE_TABLE_NAME member functions are used in the DDL LCR rule because the GET_OBJECT_OWNER and GET_OBJECT_NAME functions can return NULL if a user who does not own an object performs a DDL change on the object.

Summary of Rules In this example, the following table rules were defined:

- A table rule that evaluates to `TRUE` if a row LCR contains a row change that results from a DML operation on the `hr.locations` table.
- A table rule that evaluates to `TRUE` if a DDL LCR contains a DDL change performed on the `hr.countries` table.

Given these rules, the following list provides examples of changes applied by an apply process:

- A row is inserted into the `hr.locations` table.
- Five rows are deleted from the `hr.locations` table.
- A column is added to the `hr.countries` table.

The apply process dequeues the LCRs containing these changes from its associated queue and applies them to the database objects at the **destination database**.

Given these rules, the following list provides examples of changes that are ignored by the apply process:

- A row is inserted into the `hr.employees` table. This change is not applied because a change to the `hr.employees` table does not satisfy any of the rules.
- A row is updated in the `hr.countries` table. This change is a DML change, not a DDL change. This change is not applied because the rule on the `hr.countries` table is for DDL changes only.
- A column is added to the `hr.locations` table. This change is a DDL change, not a DML change. This change is not applied because the rule on the `hr.locations` table is for DML changes only.

Subset Rules

A **subset rule** is a special type of **table rule** for DML changes that is relevant only to a subset of the rows in a table. You can create subset rules for **capture process**es, **apply process**es, and **messaging client**s using the `ADD_SUBSET_RULES` procedure, and you can create subset rules for **propagation**s using the `ADD_SUBSET_PROPAGATION_RULES` procedure. These procedures enable you to use a condition similar to a `WHERE` clause in a `SELECT` statement to specify the following:

- That a capture process only captures a subset of the row changes resulting from DML changes to a particular table
- That a propagation only propagates a subset of the row LCRs relating to a particular table
- That an apply process only applies a subset of the row LCRs relating to a particular table
- That a messaging client only dequeues a subset of the row LCRs relating to a particular table

The `ADD_SUBSET_RULES` procedure and the `ADD_SUBSET_PROPAGATION_RULES` procedure can add subset rules to the **positive rule set** only of a **Streams client**. You cannot add subset rules to the **negative rule set** for a Streams client using these procedures.

The following sections describe subset rules in more detail:

- Subset Rules Example
- Row Migration and Subset Rules
- Subset Rules and Supplemental Logging
- Guidelines for Using Subset Rules
- Restrictions for Subset Rules

> **Note:** Capture process, propagation, and messaging client subset rules can be specified only at databases running Oracle Database 10*g*, but apply process subset rules can be specified at databases running Oracle9*i* Release 2 (9.2) or later.

Subset Rules Example

This example instructs a Streams apply process to apply a subset of row LCRs relating to the `hr.regions` table where the `region_id` is `2`. These changes originated at the `dbs1.net` **source database**.

Run the `ADD_SUBSET_RULES` procedure to create three rules:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_SUBSET_RULES(
    table_name               =>  'hr.regions',
    dml_condition            =>  'region_id=2',
    streams_type             =>  'apply',
    streams_name             =>  'apply',
    queue_name               =>  'streams_queue',
    include_tagged_lcr       =>  false,
    source_database          =>  'dbs1.net');
END;
/
```

The `ADD_SUBSET_RULES` procedure creates three rules: one for `INSERT` operations, one for `UPDATE` operations, and one for `DELETE` operations.

Here is the **rule condition** used by the insert rule:

```
:dml.get_object_owner()='HR' AND :dml.get_object_name()='REGIONS'
AND :dml.is_null_tag()='Y' AND :dml.get_source_database_name()='DBS1.NET'
AND :dml.get_command_type() IN ('UPDATE','INSERT')
AND (:dml.get_value('NEW','"REGION_ID"') IS NOT NULL)
AND (:dml.get_value('NEW','"REGION_ID"').AccessNumber()=2)
AND (:dml.get_command_type()='INSERT'
OR ((:dml.get_value('OLD','"REGION_ID"') IS NOT NULL)
AND (((:dml.get_value('OLD','"REGION_ID"').AccessNumber() IS NOT NULL)
AND NOT (:dml.get_value('OLD','"REGION_ID"').AccessNumber()=2))
OR ((:dml.get_value('OLD','"REGION_ID"').AccessNumber() IS NULL)
AND NOT EXISTS (SELECT 1 FROM SYS.DUAL
WHERE (:dml.get_value('OLD','"REGION_ID"').AccessNumber()=2))))))
```

Based on this rule condition, row LCRs are evaluated in the following ways:

- For an insert, if the new value in the row LCR for `region_id` is `2`, then the insert is applied.
- For an insert, if the new value in the row LCR for `region_id` is not `2` or is `NULL`, then the insert is filtered out.

- For an update, if the old value in the row LCR for `region_id` is not `2` or is `NULL` and the new value in the row LCR for `region_id` is `2`, then the update is converted into an insert and applied. This automatic conversion is called **row migration**. See "Row Migration and Subset Rules" on page 6-20 for more information.

Here is the rule condition used by the update rule:

```
:dml.get_object_owner()='HR' AND :dml.get_object_name()='REGIONS'
AND :dml.is_null_tag()='Y' AND :dml.get_source_database_name()='DBS1.NET'
AND :dml.get_command_type()='UPDATE'
AND (:dml.get_value('NEW','"REGION_ID"') IS NOT NULL)
AND (:dml.get_value('OLD','"REGION_ID"') IS NOT NULL)
AND (:dml.get_value('OLD','"REGION_ID"').AccessNumber()=2)
AND (:dml.get_value('NEW','"REGION_ID"').AccessNumber()=2)
```

Based on this rule condition, row LCRs are evaluated in the following ways:

- For an update, if both the old value and the new value in the row LCR for `region_id` are `2`, then the update is applied as an update.
- For an update, if either the old value or the new value in the row LCR for `region_id` is not `2` or is `NULL`, then the update does not satisfy the update rule. The LCR can satisfy the insert rule, the delete rule, or neither rule.

Here is the rule condition used by the delete rule:

```
:dml.get_object_owner()='HR' AND :dml.get_object_name()='REGIONS'
AND :dml.is_null_tag()='Y' AND :dml.get_source_database_name()='DBS1.NET'
AND :dml.get_command_type() IN ('UPDATE','DELETE')
AND (:dml.get_value('OLD','"REGION_ID"') IS NOT NULL)
AND (:dml.get_value('OLD','"REGION_ID"').AccessNumber()=2)
AND (:dml.get_command_type()='DELETE'
OR ((:dml.get_value('NEW','"REGION_ID"') IS NOT NULL)
AND (((:dml.get_value('NEW','"REGION_ID"').AccessNumber() IS NOT NULL)
AND NOT (:dml.get_value('NEW','"REGION_ID"').AccessNumber()=2))
OR ((:dml.get_value('NEW','"REGION_ID"').AccessNumber() IS NULL)
AND NOT EXISTS (SELECT 1 FROM SYS.DUAL
WHERE (:dml.get_value('NEW','"REGION_ID"').AccessNumber()=2))))))
```

Based on this rule condition, row LCRs are evaluated in the following ways:

- For a delete, if the old value in the row LCR for `region_id` is `2`, then the delete is applied.
- For a delete, if the old value in the row LCR for `region_id` is not `2` or is `NULL`, then the delete is filtered out.
- For an update, if the old value in the row LCR for `region_id` is `2` and the new value in the row LCR for `region_id` is not `2` or is `NULL`, then the update is converted into a delete and applied. This automatic conversion is called row migration. See "Row Migration and Subset Rules" on page 6-20 for more information.

Given these subset rules, the following list provides examples of changes applied by an apply process:

- A row is updated in the `hr.regions` table where the old `region_id` is `4` and the new value of `region_id` is `2`. This update is transformed into an insert.
- A row is updated in the `hr.regions` table where the old `region_id` is `2` and the new value of `region_id` is `1`. This update is transformed into a delete.

The apply process dequeues row LCRs containing these changes from its associated **queue** and applies them to the `hr.regions` table at the **destination database**.

Given these subset rules, the following list provides examples of changes that are ignored by the apply process:

- A row is inserted into the `hr.employees` table. This change is not applied because a change to the `hr.employees` table does not satisfy the subset rules.
- A row is updated in the `hr.regions` table where the `region_id` was `1` before the update and remains `1` after the update. This change is not applied because the subset rules for the `hr.regions` table evaluate to `TRUE` only when the new or old (or both) values for `region_id` is `2`.

Row Migration and Subset Rules

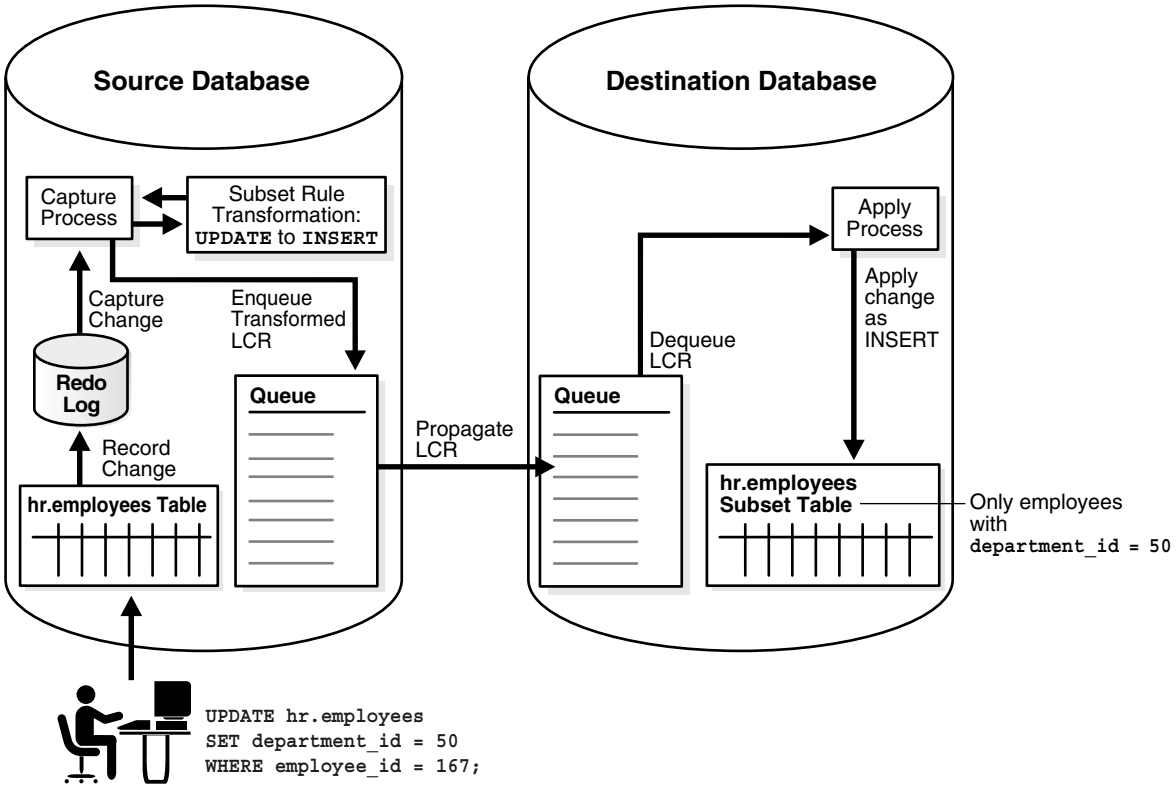
When you use subset rules, an update operation can be converted into an insert or delete operation when it is captured, propagated, applied, or dequeued. This automatic conversion is called **row migration** and is performed by an internal transformation specified automatically in the **action context** for a subset rule. The following sections describe row migration during capture, propagation, apply, and dequeue.

Attention: Subset rules should reside only in positive rule sets. Do not add subset rules to negative rule sets. Doing so can have unpredictable results, because row migration would not be performed on LCRs that are not discarded by the negative rule set. Also, row migration is not performed on LCRs discarded because they evaluate to `TRUE` against a negative rule set.

Row Migration During Capture When a subset rule is in the rule set for a **capture process**, an update that satisfies the subset rule can be converted into an insert or delete when it is captured.

For example, suppose you use a subset rule to specify that a capture process captures changes to the `hr.employees` table where the employee's `department_id` is `50` using the following subset condition: `department_id = 50`. Assume that the table at the source database contains records for employees from all departments. If a DML operation changes an employee's `department_id` from `80` to `50`, then the capture process with the subset rule converts the update operation into an insert operation and captures the change. Therefore, a row LCR that contains an `INSERT` is enqueued into the capture process queue. Figure 6–2 illustrates this example.

Figure 6–2 Row Migration During Capture

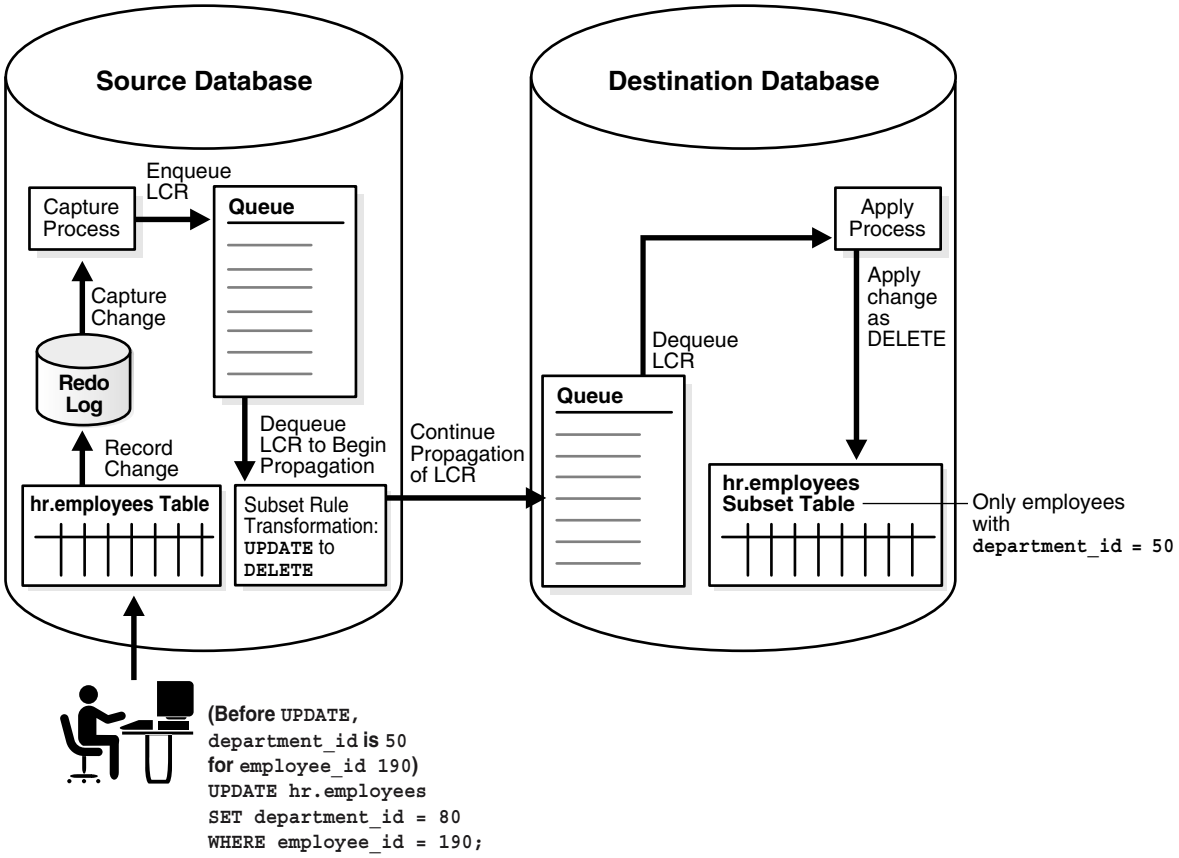


Similarly, if a captured update changes an employee's `department_id` from `50` to `20`, then a capture process with this subset rule converts the update operation into a `DELETE` operation.

Row Migration During Propagation When a subset rule is in the rule set for a propagation, an update operation can be converted into an insert or delete operation when a row LCR is propagated.

For example, suppose you use a subset rule to specify that a propagation propagates changes to the `hr.employees` table where the employee's `department_id` is `50` using the following subset condition: `department_id = 50`. If the **source queue** for the propagation contains a row LCR with an update operation on the `hr.employees` table that changes an employee's `department_id` from `50` to `80`, then the propagation with the subset rule converts the update operation into a delete operation and propagates the row LCR to the **destination queue**. Therefore, a row LCR that contains a `DELETE` is enqueued into the destination queue. Figure 6–3 illustrates this example.

Figure 6–3 Row Migration During Propagation

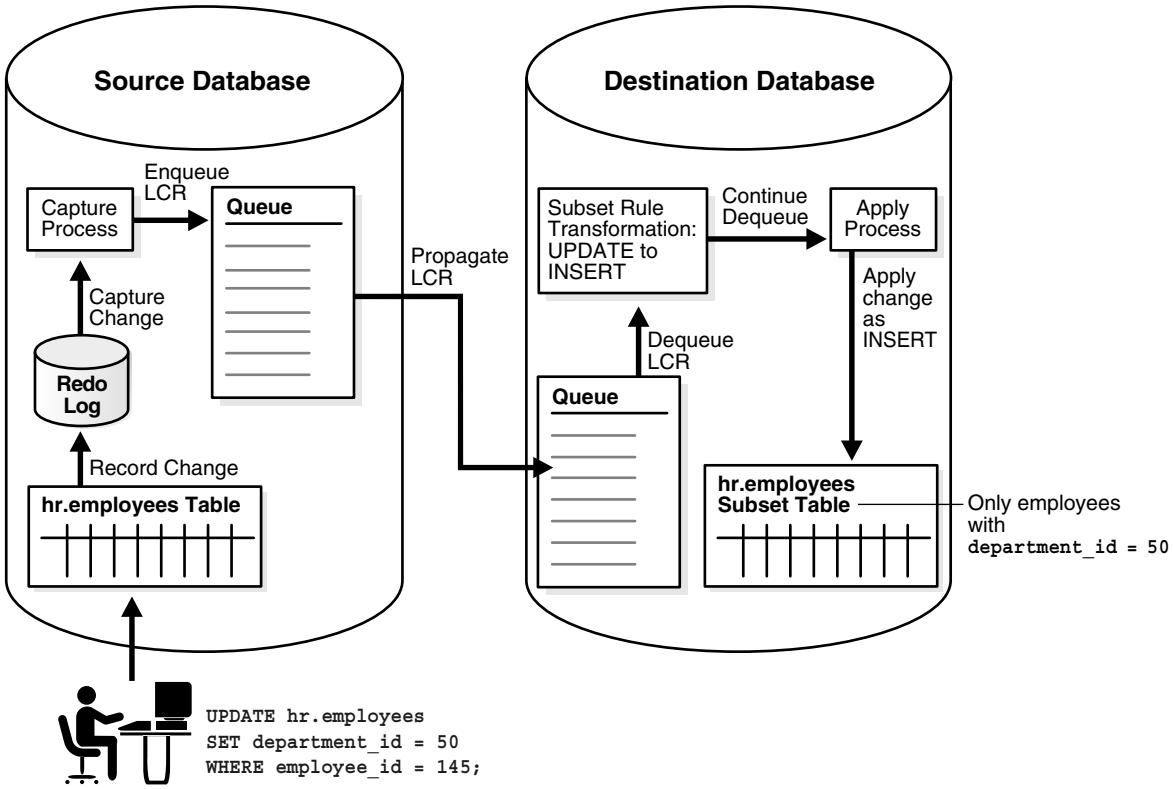


Similarly, if a captured update changes an employee's `department_id` from `80` to `50`, then a propagation with this subset rule converts the update operation into an `INSERT` operation.

Row Migration During Apply When a subset rule is in the rule set for an **apply process**, an update operation can be converted into an insert or delete operation when a row LCR is applied.

For example, suppose you use a subset rule to specify that an apply process applies changes to the `hr.employees` table where the employee's `department_id` is `50` using the following subset condition: `department_id = 50`. Assume that the table at the destination database is a subset table that only contains records for employees whose `department_id` is `50`. If a source database captures a change to an employee that changes the employee's `department_id` from `80` to `50`, then the apply process with the subset rule at a destination database applies this change by converting the update operation into an insert operation. This conversion is needed because the employee's row does not exist in the destination table. Figure 6–4 illustrates this example.

Figure 6–4 Row Migration During Apply

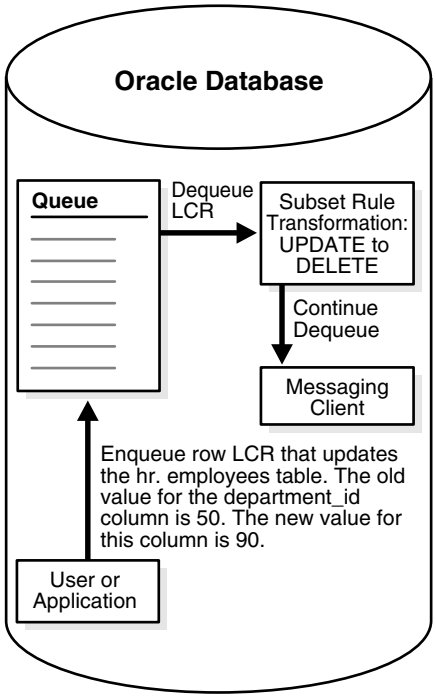


Similarly, if a captured update changes an employee's `department_id` from `50` to `20`, then an apply process with this subset rule converts the update operation into a `DELETE` operation.

Row Migration During Dequeue by a Messaging Client When a subset rule is in the rule set for a messaging client, an update operation can be converted into an insert or delete operation when a row LCR is dequeued.

For example, suppose you use a subset rule to specify that a messaging client dequeues changes to the `hr.employees` table when the employee's `department_id` is `50` using the following subset condition: `department_id = 50`. If the queue for a messaging client contains a user-enqueued row LCR with an update operation on the `hr.employees` table that changes an employee's `department_id` from `50` to `90`, then when a user or application invokes a messaging client with this subset rule, the messaging client converts the update operation into a delete operation and dequeues the row LCR. Therefore, a row LCR that contains a `DELETE` is dequeued. The messaging client can process this row LCR in any customized way. For example, it can send the row LCR to a custom application. Figure 6–5 illustrates this example.

Figure 6–5 Row Migration During Dequeue by a Messaging Client



Similarly, if a user-enqueued row LCR contains an update that changes an employee's `department_id` from `90` to `50`, then a messaging client with this subset rule converts the `UPDATE` operation into an `INSERT` operation during dequeue.

Subset Rules and Supplemental Logging

If you specify a subset rule for a table for capture, propagation, or apply, then an unconditional **supplemental log group** must be specified at the source database for all the columns in the subset condition and all of the columns in the table(s) at the destination database(s) that will apply these changes. In some cases, when a subset rule is specified, an update can be converted to an insert, and, in these cases, supplemental information might be needed for some or all of the columns.

For example, if you specify a subset rule for an apply process at database `dbs2.net` on the `postal_code` column in the `hr.locations` table, and the source database for changes to this table is `dbs1.net`, then specify **supplemental logging** at `dbs1.net` for all of the columns that exist in the `hr.locations` table at `dbs2.net`, as well as the `postal_code` column, even if this column does not exist in the table at the destination database.

> **See Also:** *Oracle Streams Replication Administrator's Guide* for detailed information about supplemental logging

Guidelines for Using Subset Rules

The following sections provide guidelines for using subset rules:

- Use Capture Subset Rules When All Destinations Need Only a Subset of Changes
- Use Propagation or Apply Subset Rules When Some Destinations Need Subsets
- Make Sure the Table Where Subset Row LCRs Are Applied Is a Subset Table

Use Capture Subset Rules When All Destinations Need Only a Subset of Changes Subset rules should be used with a capture process when all destination databases of the capture process need only row changes that satisfy the subset condition for the table. In this case, a capture process captures a subset of the DML changes to the table, and one or more propagations propagate these changes in the form of row LCRs to one or more destination databases. At each destination database, an apply process applies these row LCRs to a subset table in which all of the rows satisfy the subset condition in the subset rules for the capture process. None of the destination databases need all of the DML changes made to the table. When you use subset rules for a **local capture process**, some additional overhead is incurred to perform row migrations at the site running the source database.

Use Propagation or Apply Subset Rules When Some Destinations Need Subsets Subset rules should be used with a propagation or an apply process when some destinations in an environment need only a subset of captured DML changes. The following are examples of such an environment:

- Most of the destination databases for captured DML changes to a table need a different subset of these changes.
- Most of the destination databases need all of the captured DML changes to a table, but some destination databases need only a subset of these changes.

In these types of environments, the capture process must capture all of the changes to the table, but you can use subset rules with propagations and apply processes to ensure that subset tables at destination databases only apply the correct subset of captured DML changes.

Consider these factors when you decide to use subset rules with a propagation in this type of environment:

- You can reduce network traffic because fewer row LCRs are propagated over the network.
- The site that contains the source queue for the propagation incurs some additional overhead to perform row migrations.

Consider these factors when you decide to use subset rules with an apply process in this type of environment:

- The queue used by the apply process can contain all row LCRs for the subset table. In a **directed network**s environment, propagations can propagate any of the row LCRs for the table to destination queues as appropriate, whether or not the apply process applies these row LCRs.
- The site that is running the apply process incurs some additional overhead to perform row migrations.

Make Sure the Table Where Subset Row LCRs Are Applied Is a Subset Table If an apply process might apply row LCRs that have been transformed by a row migration, then Oracle recommends that the table at the destination database be a subset table where each row matches the condition in the subset rule. If the table is not such a subset table, then apply errors might result.

For example, consider a scenario in which a subset rule for a capture process has the condition `department_id = 50` for DML changes to the `hr.employees` table. If the `hr.employees` table at a destination database of this capture process contains rows for employees in all departments, not just in department `50`, then a constraint violation might result during apply:

1. At the source database, a DML change updates the `hr.employees` table and changes the `department_id` for the employee with an `employee_id` of `100` from `90` to `50`.
2. A capture process using the subset rule captures the change and converts the update into an insert and enqueues the change into the capture process queue as a row LCR.
3. A propagation propagates the row LCR to the destination database without modifying it.
4. An apply process attempts to apply the row LCR as an insert at the destination database, but an employee with an `employee_id` of `100` already exists in the `hr.employees` table, and an apply error results.

In this case, if the table at the destination database were a subset of the `hr.employees` table and only contained rows of employees whose `department_id` was `50`, then the insert would have been applied successfully.

Similarly, if an apply process might apply row LCRs that have been transformed by a row migration to a table, and you allow users or applications to perform DML operations on the table, then Oracle recommends that all DML changes satisfy the subset condition. If you allow local changes to the table, then the apply process cannot ensure that all rows in the table meet the subset condition. For example, suppose the condition is `department_id = 50` for the `hr.employees` table. If a user or an application inserts a row for an employee whose `department_id` is `30`, then this row remains in the table and is not removed by the apply process. Similarly, if a user or an application updates a row locally and changes the `department_id` to `30`, then this row also remains in the table.

Restrictions for Subset Rules

The following restrictions apply to subset rules:

- A table with the table name referenced in the subset rule must exist in the same database as the subset rule, and this table must be in the same schema referenced for the table in the subset rule.
- If the subset rule is in the positive rule set for a capture process, then the table must contain the columns specified in the subset condition, and the datatype of each of these columns must match the datatype of the corresponding column at the source database.
- If the subset rule is in the positive rule set for a propagation or apply process, then the table must contain the columns specified in the subset condition, and the datatype of each column must match the datatype of the corresponding column in row LCRs that evaluate to `TRUE` for the subset rule.
- Creating subset rules for tables that have one or more LOB, `LONG`, `LONG RAW`, or user-defined type columns is not supported.

Message Rules

When you use a **rule** to specify a Streams task that is relevant only for a **user-enqueued message** of a specific **message** type, you are specifying a **message rule**. You can specify message rules for **propagation**s, **apply process**es, and **messaging client**s.

A single message rule in the **positive rule set** for a propagation means that the propagation propagates the user-enqueued messages of the message type in the **source queue** that satisfy the **rule condition**. A single message rule in the **negative**

rule set for a propagation means that the propagation discards the user-enqueued messages of the message type in the source queue that satisfy the rule condition.

A single message rule in the positive rule set for an apply process means that the apply process dequeues user-enqueued messages of the message type that satisfy the rule condition. The apply process then sends these user-enqueued messages to its **message handler**. A single message rule in the negative rule set for an apply process means that the apply process discards user-enqueued messages of the message type in its **queue** that satisfy the rule condition.

A single message rule in the positive rule set for a messaging client means that a user or an application can use the messaging client to dequeue user-enqueued messages of the message type that satisfy the rule condition. A single message rule in the negative rule set for a messaging client means that the messaging client discards user-enqueued messages of the message type in its queue that satisfy the rule condition. Unlike propagations and apply processes, which propagate or apply messages automatically when they are running, a messaging client does not automatically dequeue or discard messages. Instead, a messaging client must be invoked by a user or application to dequeue or discard messages.

Message Rule Example

Suppose you use the `ADD_MESSAGE_RULE` procedure in the `DBMS_STREAMS_ADM` package to instruct a **Streams client** to behave in the following ways:

- Dequeue User-Enqueued Messages If region Is EUROPE and priority Is 1
- Send User-Enqueued Messages to a Message Handler If region Is AMERICAS and priority Is 2

The first instruction in the previous list pertains to a messaging client, while the second instruction pertains to an apply process.

The rules created in these examples are for messages of the following type:

```
CREATE TYPE strmadmin.region_pri_msg AS OBJECT(
  region         VARCHAR2(100),
  priority       NUMBER,
  message        VARCHAR2(3000))
/
```

Dequeue User-Enqueued Messages If region Is EUROPE and priority Is 1 Run the `ADD_MESSAGE_RULE` procedure to create a rule for messages of `region_pri_msg` type:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_MESSAGE_RULE (
    message_type    =>  'strmadmin.region_pri_msg',
    rule_condition  =>  ':msg.region = ''EUROPE'' AND  ' ||
                        ':msg.priority = ''1'' ',
    streams_type    =>  'dequeue',
    streams_name    =>  'msg_client',
    queue_name      =>  'streams_queue',
    inclusion_rule  =>  true);
END;
/
```

Notice that `dequeue` is specified for the `streams_type` parameter. Therefore, this procedure creates a messaging client named `msg_client` if it does not already exist. If this messaging client already exists, then this procedure adds the message rule to its rule set. Also, notice that the `inclusion_rule` parameter is set to `true`. This setting means that the **system-created rule** is added to the positive rule set for the messaging

client. The user who runs this procedure is granted the privileges to dequeue from the queue using the messaging client.

The ADD_MESSAGE_RULE procedure creates a rule with a rule condition similar to the following:

```
:"VAR$_52".region = 'EUROPE' AND  :"VAR$_52".priority = '1'
```

The variables in the rule condition that begin with VAR$ are variables that are specified in the system-generated **evaluation context** for the rule.

See Also: "Evaluation Contexts Used in Streams" on page 6-34

Send User-Enqueued Messages to a Message Handler If region Is AMERICAS and priority Is 2
Run the ADD_MESSAGE_RULE procedure to create a rule for messages of region_pri_msg type:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_MESSAGE_RULE (
    message_type    =>  'strmadmin.region_pri_msg',
    rule_condition  =>  ':msg.region = ''AMERICAS'' AND  ' ||
                        ':msg.priority = ''2'' ',
    streams_type    =>  'apply',
    streams_name    =>  'apply_msg',
    queue_name      =>  'streams_queue',
    inclusion_rule  =>  true);
END;
/
```

Notice that apply is specified for the streams_type parameter. Therefore, this procedure creates an apply process named apply_msg if it does not already exist. If this apply process already exists, then this procedure adds the message rule to its rule set. Also, notice that the inclusion_rule parameter is set to true. This setting means that the system-created rule is added to the positive rule set for the messaging client.

The ADD_MESSAGE_RULE procedure creates a rule with a rule condition similar to the following:

```
:"VAR$_56".region = 'AMERICAS' AND  :"VAR$_56".priority = '2'
```

The variables in the rule condition that begin with VAR$ are variables that are specified in the system-generated evaluation context for the rule.

See Also: "Evaluation Contexts Used in Streams" on page 6-34

Summary of Rules In this example, the following message rules were defined:

- A message rule for a messaging client named msg_client that evaluates to TRUE if a message has EUROPE for its region and 1 for its priority. Given this rule, a user or application can use the messaging client to dequeue messages of region_pri_msg type that satisfy the rule condition.
- A message rule for an apply process named apply_msg that evaluates to TRUE if a message has AMERICAS for its region and 2 for its priority. Given this rule, the apply process dequeues messages of region_pri_msg type that satisfy the rule condition and sends these messages to its message handler or reenqueues the messages into a specified queue.

See Also:

- "Non-LCR User Message Processing" on page 4-5
- "Enqueue Destinations for Messages During Apply" on page 6-39

System-Created Rules and Negative Rule Sets

You add system-created **rule**s to a **negative rule set** to specify that you do not want a **Streams client** to perform its task for changes that satisfy these rules. Specifically, a **system-created rule** in a negative rule set means the following for each type of Streams client:

- A **capture process** discards changes that satisfy the rule.
- A **propagation** discards **message**s in its **source queue** that satisfy the rule.
- An **apply process** discards messages in its **queue** that satisfy the rule.
- A **messaging client** discards messages in its queue that satisfy the rule.

If a Streams client does not have a negative rule set, then you can create a negative rule set and add rules to it by running one of the following procedures and setting the `inclusion_rule` parameter to `false`:

- `DBMS_STREAMS_ADM.ADD_TABLE_RULES`
- `DBMS_STREAMS_ADM.ADD_SCHEMA_RULES`
- `DBMS_STREAMS_ADM.ADD_GLOBAL_RULES`
- `DBMS_STREAMS_ADM.ADD_MESSAGE_RULE`
- `DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES`
- `DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES`
- `DBMS_STREAMS_ADM.ADD_GLOBAL_PROPAGATION_RULES`
- `DBMS_STREAMS_ADM.ADD_MESSAGE_PROPAGATION_RULE`

If a negative rule set already exists for the Streams client when you run one of these procedures, then the procedure adds the system-created rules to the existing negative rule set.

Alternatively, you can create a negative rule set when you create a Streams client by running one of the following procedures and specifying a non-`NULL` value for the `negative_rule_set_name` parameter:

- `DBMS_CAPTURE_ADM.CREATE_CAPTURE`
- `DBMS_PROPAGATION_ADM.CREATE_PROPAGATION`
- `DBMS_APPLY_ADM.CREATE_APPLY`

Also, you can specify a negative rule set for an existing Streams client by altering the client. For example, to specify a negative rule set for an existing capture process, use the `DBMS_CAPTURE_ADM.ALTER_CAPTURE` procedure. After a Streams client has a negative rule set, you can use the procedures in the `DBMS_STREAM_ADM` package listed previously to add system-created rules to it.

Instead of adding rules to a negative rule set, you can also exclude changes to certain tables or schemas in the following ways:

- Do not add system-created rules for the table or schema to a **positive rule set** for a Streams client. For example, to capture DML changes to all of the tables in a particular schema except for one table, add a DML **table rule** for each table in the schema, except for the excluded table, to the positive rule set for the capture process. The disadvantages of this approach are that there can be many tables in a schema and each one requires a separate DML rule, and, if a new table is added to the schema, and you want to capture changes to this new table, then a new DML rule must be added for this table to the positive rule set for the capture process.
- Use the `NOT` logical condition in the **rule condition** of a complex rule in the positive rule set for a Streams client. For example, to capture DML changes to all of the tables in a particular schema except for one table, use the `DBMS_STREAMS_ADM.ADD_SCHEMA_RULES` procedure to add a system-created DML **schema rule** to the positive rule set for the capture process that instructs the capture process to capture changes to the schema, and use the `and_condition` parameter to exclude the table with the `NOT` logical condition. The disadvantages to this approach are that it involves manually specifying parts of rule conditions, which can be error prone, and rule evaluation is not as efficient for complex rules as it is for unmodified system-created rules.

Given the goal of capturing DML changes to all of the tables in a particular schema except for one table, you can add a DML schema rule to the positive rule set for the capture process and a DML table rule for the excluded table to the negative rule set for the capture process.

This approach has the following advantages over the alternatives described previously:

- You add only two rules to achieve the goal.
- If a new table is added to the schema, and you want to capture DML changes to the table, then the capture process captures these changes without requiring modifications to existing rules or additions of new rules.
- You do not need to specify or edit rule conditions manually.
- Rule evaluation is more efficient because you avoid using complex rules.

See Also:

- "Complex Rule Conditions" on page 6-43
- "System-Created Rules with Added User-Defined Conditions" on page 6-32 for more information about the `and_condition` parameter

Negative Rule Set Example

Suppose you want to apply row LCRs that contain the results of DML changes to all of the tables in `hr` schema except for the `job_history` table. To do so, you can use the `ADD_SCHEMA_RULES` procedure in the `DBMS_STREAMS_ADM` package to instruct a Streams apply process to apply row LCRs that contain the results of DML changes to the tables in the `hr` schema. In this case, the procedure creates a schema rule and adds the rule to the positive rule set for the apply process.

You can use the `ADD_TABLE_RULES` procedure in the `DBMS_STREAMS_ADM` package to instruct the Streams apply process to discard row LCRs that contain the results of

DML changes to the tables in the hr.job_history table. In this case, the procedure creates a table rule and adds the rule to the negative rule set for the apply process.

The following sections explain how to run these procedures:

- Apply All DML Changes to the Tables in the hr Schema
- Discard Row LCRs Containing DML Changes to the hr.job_history Table

Apply All DML Changes to the Tables in the hr Schema These changes originated at the dbs1.net **source database**.

Run the ADD_SCHEMA_RULES procedure to create this rule:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
    schema_name         =>  'hr',
    streams_type        =>  'apply',
    streams_name        =>  'apply',
    queue_name          =>  'streams_queue',
    include_dml         =>  true,
    include_ddl         =>  false,
    include_tagged_lcr  =>  false,
    source_database     =>  'dbs1.net',
    inclusion_rule      =>  true);
END;
/
```

Notice that the inclusion_rule parameter is set to true. This setting means that the system-created rule is added to the positive rule set for the apply process.

The ADD_SCHEMA_RULES procedure creates a rule with a rule condition similar to the following:

```
((:dml.get_object_owner() = 'HR') and :dml.is_null_tag() = 'Y'
and :dml.get_source_database_name() = 'DBS1.NET' )
```

Discard Row LCRs Containing DML Changes to the hr.job_history Table These changes originated at the dbs1.net source database.

Run the ADD_TABLE_RULES procedure to create this rule:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name          =>  'hr.job_history',
    streams_type        =>  'apply',
    streams_name        =>  'apply',
    queue_name          =>  'streams_queue',
    include_dml         =>  true,
    include_ddl         =>  false,
    include_tagged_lcr  =>  true,
    source_database     =>  'dbs1.net',
    inclusion_rule      =>  false);
END;
/
```

Notice that the inclusion_rule parameter is set to false. This setting means that the system-created rule is added to the negative rule set for the apply process.

Also notice that the include_tagged_lcr parameter is set to true. This setting means that all changes for the table, including tagged LCRs that satisfy all of the other rule conditions, will be discarded. In most cases, specify true for the include_tagged_lcr parameter if the inclusion_rule parameter is set to false.

The ADD_TABLE_RULES procedure creates a rule with a rule condition similar to the following:

```
(((:dml.get_object_owner() = 'HR' and :dml.get_object_name() = 'JOB_HISTORY'))
and :dml.get_source_database_name() = 'DBS1.NET' )
```

Summary of Rules In this example, the following rules were defined:

- A schema rule that evaluates to TRUE if a DML operation is performed on the tables in the hr schema. This rule is in the positive rule set for the apply process.
- A table rule that evaluates to TRUE if a DML operation is performed on the hr.job_history table. This rule is in the negative rule set for the apply process.

Given these rules, the following list provides examples of changes applied by the apply process:

- A row is inserted into the hr.departments table.
- Five rows are updated in the hr.employees table.
- A row is deleted from the hr.countries table.

The apply process dequeues these changes from its associated queue and applies them to the database objects at the destination database.

Given these rules, the following list provides examples of changes that are ignored by the apply process:

- A row is inserted into the hr.job_history table.
- A row is updated in the hr.job_history table.
- A row is deleted from the hr.job_history table.

These changes are not applied because they satisfy a rule in the negative rule set for the apply process.

> **See Also:** "Rule Sets and Rule Evaluation of Messages" on page 6-3

System-Created Rules with Added User-Defined Conditions

Some of the procedures that create **rule**s in the DBMS_STREAMS_ADM package include an and_condition parameter. This parameter enables you to add conditions to **system-created rule**s. The condition specified by the and_condition parameter is appended to the system-created **rule condition** using an AND clause in the following way:

```
(system_condition) AND (and_condition)
```

The variable in the specified condition must be :lcr. For example, to specify that the **table rule**s generated by the ADD_TABLE_RULES procedure evaluate to TRUE only if the table is hr.departments, the **source database** is dbs1.net, and the Streams **tag** is the hexadecimal equivalent of '02', run the following procedure:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name          =>  'hr.departments',
    streams_type        =>  'apply',
    streams_name        =>  'apply_02',
    queue_name          =>  'streams_queue',
    include_dml         =>  true,
    include_ddl         =>  true,
    include_tagged_lcr  =>  true,
    source_database     =>  'dbs1.net',
    inclusion_rule      =>  true,
    and_condition       =>  ':lcr.get_tag() = HEXTORAW(''02'')');
END;
/
```

The `ADD_TABLE_RULES` procedure creates a DML rule with the following condition:

```
(((((:dml.get_object_owner() = 'HR' and :dml.get_object_name() = 'DEPARTMENTS'))
 and :dml.get_source_database_name() = 'DBS1.NET' ))
and (:dml.get_tag() = HEXTORAW('02')))
```

It creates a DDL rule with the following condition:

```
(((((:ddl.get_object_owner() = 'HR' and :ddl.get_object_name() = 'DEPARTMENTS')
or (:ddl.get_base_table_owner() = 'HR'
and :ddl.get_base_table_name() = 'DEPARTMENTS'))
and :ddl.get_source_database_name() = 'DBS1.NET' ))
and (:ddl.get_tag() = HEXTORAW('02')))
```

Notice that the `:lcr` in the specified condition is converted to `:dml` or `:ddl`, depending on the rule that is being generated. If you are specifying an LCR member subprogram that is dependent on the LCR type (row or DDL), then make sure this procedure only generates the appropriate rule. Specifically, if you specify an LCR member subprogram that is valid only for row LCRs, then specify `true` for the `include_dml` parameter and `false` for the `include_ddl` parameter. If you specify an LCR member subprogram that is valid only for DDL LCRs, then specify `false` for the `include_dml` parameter and `true` for the `include_ddl` parameter.

For example, the `GET_OBJECT_TYPE` member function only applies to DDL LCRs. Therefore, if you use this member function in an `and_condition`, then specify `false` for the `include_dml` parameter and `true` for the `include_ddl` parameter.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for more information about LCR member subprograms
- *Oracle Streams Replication Administrator's Guide* for more information about Streams tags

Evaluation Contexts Used in Streams

The following sections describe the system-created **evaluation context**s used in Streams.

- Evaluation Context for Global, Schema, Table, and Subset Rules
- Evaluation Contexts for Message Rules

Evaluation Context for Global, Schema, Table, and Subset Rules

When you create global, schema, table, and subset rules, the system-created **rule set**s and rules use a built-in **evaluation context** in the SYS schema named STREAMS$_EVALUATION_CONTEXT. PUBLIC is granted the EXECUTE privilege on this evaluation context. Global, schema, table, and subset rules can be used by **capture process**es, **propagation**s, **apply process**es, and **messaging client**s.

During Oracle installation, the following statement creates the Streams evaluation context:

```
DECLARE
  vt  SYS.RE$VARIABLE_TYPE_LIST;
BEGIN
  vt := SYS.RE$VARIABLE_TYPE_LIST(
    SYS.RE$VARIABLE_TYPE('DML', 'SYS.LCR$_ROW_RECORD',
       'SYS.DBMS_STREAMS_INTERNAL.ROW_VARIABLE_VALUE_FUNCTION',
       'SYS.DBMS_STREAMS_INTERNAL.ROW_FAST_EVALUATION_FUNCTION'),
    SYS.RE$VARIABLE_TYPE('DDL', 'SYS.LCR$_DDL_RECORD',
       'SYS.DBMS_STREAMS_INTERNAL.DDL_VARIABLE_VALUE_FUNCTION',
       'SYS.DBMS_STREAMS_INTERNAL.DDL_FAST_EVALUATION_FUNCTION'));
    SYS.RE$VARIABLE_TYPE(NULL, 'SYS.ANYDATA',
       NULL,
       'SYS.DBMS_STREAMS_INTERNAL.ANYDATA_FAST_EVAL_FUNCTION'));
  DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT(
    evaluation_context_name => 'SYS.STREAMS$_EVALUATION_CONTEXT',
    variable_types          => vt,
    evaluation_function     =>
                        'SYS.DBMS_STREAMS_INTERNAL.EVALUATION_CONTEXT_FUNCTION');
END;
/
```

This statement includes references to the following internal functions in the SYS.DBMS_STREAM_INTERNAL package:

- ROW_VARIABLE_VALUE_FUNCTION
- DDL_VARIABLE_VALUE_FUNCTION
- EVALUATION_CONTEXT_FUNCTION
- ROW_FAST_EVALUATION_FUNCTION
- DDL_FAST_EVALUATION_FUNCTION
- ANYDATA_FAST_EVAL_FUNCTION

> **Attention:** Information about these internal functions is provided for reference purposes only. You should never run any of these functions directly.

The ROW_VARIABLE_VALUE_FUNCTION converts an ANYDATA payload, which encapsulates a SYS.LCR$_ROW_RECORD instance, into a SYS.LCR$_ROW_RECORD instance prior to evaluating rules on the data.

The DDL_VARIABLE_VALUE_FUNCTION converts an ANYDATA payload, which encapsulates a SYS.LCR$_DDL_RECORD instance, into a SYS.LCR$_DDL_RECORD instance prior to evaluating rules on the data.

The EVALUATION_CONTEXT_FUNCTION is specified as an evaluation_function in the call to the CREATE_EVALUATION_CONTEXT procedure. This function supplements normal rule evaluation for **captured message**s. A capture process enqueues row LCRs and DDL LCRs into its **queue**, and this function enables it to enqueue other internal **message**s into the queue, such as commits, rollbacks, and data dictionary changes. This information is also used during rule evaluation for a propagation or apply process.

ROW_FAST_EVALUATION_FUNCTION improves performance by optimizing access to the following LCR$_ROW_RECORD member functions during rule evaluation:

- GET_OBJECT_OWNER
- GET_OBJECT_NAME
- IS_NULL_TAG
- GET_SOURCE_DATABASE_NAME
- GET_COMMAND_TYPE

DDL_FAST_EVALUATION_FUNCTION improves performance by optimizing access to the following LCR$_DDL_RECORD member functions during rule evaluation if the condition is <, <=, =, >=, or > and the other operand is a constant:

- GET_OBJECT_OWNER
- GET_OBJECT_NAME
- IS_NULL_TAG
- GET_SOURCE_DATABASE_NAME
- GET_COMMAND_TYPE
- GET_BASE_TABLE_NAME
- GET_BASE_TABLE_OWNER

ANYDATA_FAST_EVAL_FUNCTION improves performance by optimizing access to values inside an ANYDATA object.

Rules created using the DBMS_STREAMS_ADM package use ROW_FAST_EVALUATION_FUNCTION or DDL_FAST_EVALUATION_FUNCTION, except for subset rules created using the ADD_SUBSET_RULES or ADD_SUBSET_PROPAGATION_RULES procedure.

> **See Also:** *Oracle Database PL/SQL Packages and Types Reference* for more information about LCRs and their member functions

Evaluation Contexts for Message Rules

When you use either the ADD_MESSAGE_RULE procedure or the ADD_MESSAGE_PROPAGATION_RULE procedure to create a **message rule**, the message rule uses a user-defined **message** type that you specify when you create the rule. Such a system-created message rule uses a system-created **evaluation context**. The name of the system-created evaluation context is different for each message type used to create message rules. Such an evaluation context has a system-generated name and is created

in the schema that owns the rule. Only the user who owns this evaluation context is granted the EXECUTE privilege on it.

The evaluation context for this type of message rule contains a variable that is the same type as the message type. The name of this variable is in the form VAR$_*number*, where *number* is a system-generated number. For example, if you specify strmadmin.region_pri_msg as the message type when you create a message rule, then the system-created evaluation context has a variable of this type, and the variable is used in the **rule condition**. Assume that the following statement created the strmadmin.region_pri_msg type:

```
CREATE TYPE strmadmin.region_pri_msg AS OBJECT(
  region         VARCHAR2(100),
  priority       NUMBER,
  message        VARCHAR2(3000))
/
```

When you create a message rule using this type, you can specify the following rule condition:

```
:msg.region = 'EUROPE' AND :msg.priority = '1'
```

The system-created message rule replaces :msg in the rule condition you specify with the name of the variable. The following is an example of a message rule condition that might result:

```
:VAR$_52.region = 'EUROPE' AND  :VAR$_52.priority = '1'
```

In this case, VAR$_52 is the variable name, the type of the VAR$_52 variable is strmadmin.region_pri_msg, and the evaluation context for the rule contains this variable.

The message rule itself has an evaluation context. A statement similar to the following creates an evaluation context for a message rule:

```
DECLARE
  vt  SYS.RE$VARIABLE_TYPE_LIST;
BEGIN
  vt := SYS.RE$VARIABLE_TYPE_LIST(
    SYS.RE$VARIABLE_TYPE('VAR$_52', 'STRMADMIN.REGION_PRI_MSG',
       'SYS.DBMS_STREAMS_INTERNAL.MSG_VARIABLE_VALUE_FUNCTION', NULL));
  DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT(
    evaluation_context_name => 'STRMADMIN.EVAL_CTX$_99',
    variable_types          => vt,
    evaluation_function     => NULL);
END;
/
```

The name of the evaluation context is in the form EVAL_CTX$_*number*, where *number* is a system-generated number. In this example, the name of the evaluation context is EVAL_CTX$_99.

This statement also includes a reference to the MSG_VARIABLE_VALUE_FUNCTION internal function in the SYS.DBMS_STREAM_INTERNAL package. This function converts an ANYDATA payload, which encapsulates a message instance, into an instance of the same type as the variable prior to evaluating rules on the data. For example, if the variable type is strmadmin.region_pri_msg, then the MSG_VARIABLE_VALUE_FUNCTION converts the message payload from an ANYDATA payload to a strmadmin.region_pri_msg payload.

If you create rules for different message types, then Oracle creates a different evaluation context for each message type. If you create a new rule with the same message type as an existing rule, then the new rule uses the evaluation context for the existing rule. When you use the `ADD_MESSAGE_RULE` or `ADD_MESSAGE_PROPAGATION_RULE` to create a **rule set** for a **messaging client** or **apply process**, the new rule set does not have an evaluation context.

See Also:

- "Message Rules" on page 6-26
- "Evaluation Context for Global, Schema, Table, and Subset Rules" on page 6-34

Streams and Event Contexts

In Streams, **capture process**es and **messaging client**s do not use event contexts, but **propagation**s and **apply process**es do. Both **captured message**s and **user-enqueued message**s can be staged in a **queue**. When a **message** is staged in a queue, a propagation or apply process can send the message, along with an event context, to the **rules engine** for evaluation. An event context always has the following name-value pair: `AQ$_MESSAGE` as the name and the message as the value.

If you create a custom **evaluation context**, then you can create propagation and apply process **rule**s that refer to Streams events using implicit variables. The variable value function for each implicit variable can check for event contexts with the name `AQ$_MESSAGE`. If an event context with this name is found, then the variable value function returns a value based on a message. You can also pass the event context to an evaluation function and a variable method function.

See Also:

- "Rule Set Evaluation" on page 5-10 for more information about event contexts
- "Explicit and Implicit Variables" on page 5-6 for more information about variable value functions
- "Evaluation Function" on page 5-7

Streams and Action Contexts

The following sections describe the purposes of **action context**s in Streams and the importance of ensuring that only one **rule** in a **rule set** can evaluate to `TRUE` for a particular **rule condition**.

Purposes of Action Contexts in Streams

In Streams, an **action context** serves the following purposes:

- Internal LCR Transformations in Subset Rules
- Information About Declarative Rule-Based Transformations
- Custom Rule-Based Transformations
- Enqueue Destinations for Messages During Apply
- Execution Directives for Messages During Apply

A different name-value pair can exist in the action context of a **rule** for each of these purposes. If an action context for a rule contains more than one of these name-value pairs, then the actions specified or described by the name-value pairs are performed in the following order:

1. Perform subset transformation
2. Display information about **declarative rule-based transformation**
3. Perform **custom rule-based transformation**
4. Follow execution directive and perform execution if directed to do so (apply only)
5. Enqueue into a **destination queue** (apply only)

Note: The actions specified in the action context for a rule are performed only if the rule is in the **positive rule set** for a **capture process**, **propagation**, **apply process**, or **messaging client**. If a rule is in a **negative rule set**, then these **Streams client**s ignore the action context of the rule.

Internal LCR Transformations in Subset Rules

When you use **subset rule**s, an update operation can be converted into an insert or delete operation when it is captured, propagated, applied, or dequeued. This automatic conversion is called **row migration** and is performed by an internal transformation specified in the action context when the subset rule evaluates to `TRUE`. The name-value pair for a subset transformation has `STREAMS$_ROW_SUBSET` for the name and either `INSERT` or `DELETE` for the value.

See Also:

- "Subset Rules" on page 6-17
- Chapter 15, "Managing Rule-Based Transformations" for information about using rule-based transformation with subset rules

Information About Declarative Rule-Based Transformations

A declarative rule-based transformation is an internal modification of a row LCR that results when a rule evaluates to `TRUE`. The name-value pair for a declarative rule-based transformation has `STREAMS$_INTERNAL_TRANFORM` for the name and the name of a data dictionary view that provides additional information about the transformation for the value.

The name-value pair added for a declarative rule-based transformation is for information purposes only. These name-value pairs are not used by Streams clients. However, the declarative rule-based transformations described in an action context are performed internally before any custom rule-based transformations specified in the same action context.

See Also:

- "Declarative Rule-Based Transformations" on page 7-1
- "Managing Declarative Rule-Based Transformations" on page 15-1

Custom Rule-Based Transformations

A custom rule-based transformation is any modification made by a user-defined function to a **message** when a rule evaluates to TRUE. The name-value pair for a custom rule-based transformation has STREAMS$_TRANSFORM_FUNCTION for the name and the name of the transformation function for the value.

See Also:

- "Custom Rule-Based Transformations" on page 7-2
- "Managing Custom Rule-Based Transformations" on page 15-5

Execution Directives for Messages During Apply

The SET_EXECUTE procedure in the DBMS_APPLY_ADM package specifies whether a message that satisfies the specified rule is executed by an apply process. The name-value pair for an execution directive has APPLY$_EXECUTE for the name and NO for the value if the apply process should not execute the message. If a message that satisfies a rule should be executed by an apply process, then this name-value pair is not present in the action context of the rule.

See Also: "Specifying Execute Directives for Apply Processes" on page 13-16

Enqueue Destinations for Messages During Apply

The SET_ENQUEUE_DESTINATION procedure in the DBMS_APPLY_ADM package sets the **queue** where a message that satisfies the specified rule is enqueued automatically by an apply process. The name-value pair for an enqueue destination has APPLY$_ENQUEUE for the name and the name of the destination queue for the value.

See Also: "Specifying Message Enqueues by Apply Processes" on page 13-15

Make Sure Only One Rule Can Evaluate to TRUE for a Particular Rule Condition

If you use a non-NULL **action context** for one or more **rule**s in a **positive rule set**, then make sure only one rule can evaluate to TRUE for a particular **rule condition**. If more than one rule evaluates to TRUE for a particular condition, then only one of the rules is returned, which can lead to unpredictable results.

For example, suppose two rules evaluate to TRUE if an LCR contains a DML change to the hr.employees table. The first rule has a NULL action context. The second rule has an action context that specifies a **custom rule-based transformation**. If there is a DML change to the hr.employees table, then both rules evaluate to TRUE for the change, but only one rule is returned. In this case, the transformation might or might not occur, depending on which rule is returned.

You might want to ensure that only one rule in a positive rule set can evaluate to TRUE for any condition, regardless of whether any of the rules have a non-NULL action context. By following this guideline, you can avoid unpredictable results if, for example, a non-NULL action context is added to a rule in the future.

See Also: Chapter 7, "Rule-Based Transformations"

Action Context Considerations for Schema and Global Rules

If you use an **action context** for a **custom rule-based transformation**, enqueue destination, or execute directive with a **schema rule** or **global rule**, then the action specified by the action context is carried out on a **message** if the message causes the schema or global rule to evaluate to `TRUE`. For example, if a **schema rule** has an action context that specifies a custom rule-based transformation, then the transformation is performed on LCRs for the tables in the schema.

You might want to use an action context with a schema or global rule but exclude a subset of LCRs from the action performed by the action context. For example, if you want to perform a custom rule-based transformation on all of the tables in the `hr` schema except for the `job_history` table, then make sure the transformation function returns the original LCR if the table is `job_history`.

If you want to set an enqueue destination or an execute directive for all of the tables in the `hr` schema except for the `job_history` table, then you can use a schema rule and add the following condition to it:

```
:dml.get_object_name() != 'JOB_HISTORY'
```

In this case, if you want LCRs for the `job_history` table to evaluate to `TRUE`, but you do not want to perform the enqueue or execute directive, then you can add a **table rule** for the table to a **positive rule set**. That is, the schema rule would have the enqueue destination or execute directive, but the table rule would not.

> **See Also:** "System-Created Rules" on page 6-5 for more information about schema and global rules

User-Created Rules, Rule Sets, and Evaluation Contexts

The `DBMS_STREAMS_ADM` package generates system-created **rule**s and **rule set**s, and it can specify an Oracle supplied **evaluation context** for rules and rule sets or generate system-created evaluation contexts. If you need to create rules, rule sets, or evaluation contexts that cannot be created using the `DBMS_STREAMS_ADM` package, then you can use the `DBMS_RULE_ADM` package to create them.

Use the `DBMS_RULE_ADM` package for the following reasons:

- You need to create rules with **rule condition**s that cannot be created using the `DBMS_STREAMS_ADM` package, such as rule conditions for specific types of operations, or rule conditions that use the `LIKE` condition.
- You need to create custom evaluation contexts for the rules in your Streams environment.

You can create a rule set using the `DBMS_RULE_ADM` package, and you can associate it with a **capture process**, **propagation**, **apply process**, or **messaging client**. Such a rule set can be a **positive rule set** or **negative rule set** for a **Streams client**, and a rule set can be a positive rule set for one Streams client and a negative rule set for another.

This section contains the following topics:

See Also:

- "Specifying a Rule Set for a Capture Process" on page 11-25
- "Specifying the Rule Set for a Propagation" on page 12-11
- "Specifying the Rule Set for an Apply Process" on page 13-8

User-Created Rules and Rule Sets

The following sections describe some of the types of **rule**s and **rule set**s that you can create using the DBMS_RULE_ADM package:

- Rule Conditions for Specific Types of Operations
- Rule Conditions that Instruct Streams Clients to Discard Unsupported LCRs
- Complex Rule Conditions
- Rule Conditions with Undefined Variables that Evaluate to NULL
- Variables as Function Parameters in Rule Conditions

Note: You can add user-defined conditions to a **system-created rule** by using the and_condition parameter that is available in some of the procedures in the DBMS_STREAMS_ADM package. Using the and_condition parameter is sometimes easier than creating rules with the DBMS_RULE_ADM package.

See Also: "System-Created Rules with Added User-Defined Conditions" on page 6-32 for more information about the and_condition parameter

Rule Conditions for Specific Types of Operations

In some cases, you might want to capture, propagate, apply, or dequeue only changes that contain specific types of operations. For example, you might want to apply changes containing only insert operations for a particular table, but not other operations, such as update and delete.

Suppose you want to specify a **rule condition** that evaluates to TRUE only for INSERT operations on the hr.employees table. You can accomplish this by specifying the INSERT command type in the rule condition:

```
:dml.get_command_type() = 'INSERT' AND :dml.get_object_owner() = 'HR'
AND :dml.get_object_name() = 'EMPLOYEES' AND :dml.is_null_tag() = 'Y'
```

Similarly, suppose you want to specify a rule condition that evaluates to TRUE for all DML operations on the hr.departments table, except DELETE operations. You can accomplish this by specifying the following rule condition:

```
:dml.get_object_owner() = 'HR' AND :dml.get_object_name() = 'DEPARTMENTS' AND
:dml.is_null_tag() = 'Y' AND (:dml.get_command_type() = 'INSERT' OR
:dml.get_command_type() = 'UPDATE')
```

This rule condition evaluates to TRUE for INSERT and UPDATE operations on the hr.departments table, but not for DELETE operations. Because the hr.departments table does not include any LOB columns, you do not need to specify the LOB command types for DML operations (LOB ERASE, LOB WRITE, and

LOB TRIM), but these command types should be specified in such a rule condition for a table that contains one or more LOB columns.

The following rule condition accomplishes the same behavior for the hr.departments table. That is, the following rule condition evaluates to TRUE for all DML operations on the hr.departments table, except DELETE operations:

```
:dml.get_object_owner() = 'HR' AND :dml.get_object_name() = 'DEPARTMENTS' AND
:dml.is_null_tag() = 'Y' AND :dml.get_command_type() != 'DELETE'
```

The example rule conditions described previously in this section are all simple rule conditions. However, when you add custom conditions to system-created rule conditions, the entire condition might not be a simple rule condition, and nonsimple rules might not evaluate efficiently. In general, you should use simple rule conditions whenever possible to improve rule evaluation performance. Rule conditions created using the DBMS_STREAMS_ADM package, without custom conditions added, are always simple.

See Also:

- "Simple Rule Conditions" on page 5-3
- "Complex Rule Conditions" on page 6-43

Rule Conditions that Instruct Streams Clients to Discard Unsupported LCRs

You can use the following functions in rule conditions to instruct a **Streams client** to discard LCRs that encapsulate unsupported changes:

- The GET_COMPATIBLE member function for LCRs. This function returns the minimal database compatibility required to support an LCR.
- The COMPATIBLE_9_2 function, COMPATIBLE_10_1 function, and COMPATIBLE_10_2 function in the DBMS_STREAMS package. These functions return constant values that correspond to 9.2.0, 10.1.0, and 10.2.0 compatibility in a database, respectively. You control the compatibility of an Oracle database using the COMPATIBLE initialization parameter.

For example, consider the following rule:

```
BEGIN
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name => 'strmadmin.dml_compat_9_2',
    condition => ':dml.GET_COMPATIBLE() > DBMS_STREAMS.COMPATIBLE_9_2()');
END;
/
```

If this rule is in the **negative rule set** for a Streams client, such as a **capture process**, a **propagation**, or an **apply process**, then the Streams client discards any row LCR that is not compatible with Oracle9*i* Database Release 2 (9.2).

The following is an example that is more appropriate for a **positive rule set**:

```
BEGIN
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name => 'strmadmin.dml_compat_9_2',
    condition => ':dml.GET_COMPATIBLE() <= DBMS_STREAMS.COMPATIBLE_10_1()');
END;
/
```

If this rule is in the positive rule set for a Streams client, then the Streams client discards any row LCR that is not compatible with Oracle Database 10*g* Release 1 or

earlier. That is, the Streams client processes any row LCR that is compatible with Oracle9*i* Database Release 2 (9.2) or Oracle Database 10*g* Release 1 (10.1) and satisfies the other rules in its rule sets, but it discards any row LCR that is not compatible with these releases.

Both of the rules in the previous examples evaluate efficiently. If you use **schema rule**s or **global rule**s created by the `DBMS_STREAMS_ADM` package to capture, propagate, apply, or dequeue LCRs, then rules such as these can be used to discard LCRs that are not supported by a particular database.

Note:

- You can determine which database objects in a database are not supported by Streams by querying the `DBA_STREAMS_UNSUPPORTED` data dictionary view.
- Instead of using the `DBMS_RULE_ADM` package to create rules with `GET_COMPATIBLE` conditions, you can use one of the procedures in the `DBMS_STREAMS_ADM` package to create such rules by specifying the `GET_COMPATIBLE` condition in the `AND_CONDITION` parameter.
- DDL LCRs always return `DBMS_STREAMS.COMPATIBLE_9_2`.

See Also:

- "Monitoring Compatibility in a Streams Environment" on page 26-7
- "Global Rules Example" on page 6-11, "Schema Rule Example" on page 6-14, and "System-Created Rules with Added User-Defined Conditions" on page 6-32
- *Oracle Database Reference* and *Oracle Database Upgrade Guide* for more information about the `COMPATIBLE` initialization parameter

Complex Rule Conditions

Complex rule conditions are rule conditions that do not meet the requirements for simple rule conditions described in "Simple Rule Conditions" on page 5-3. In a Streams environment, the `DBMS_STREAMS_ADM` package creates rules with simple rule conditions only, assuming no custom conditions are added to the system-created rules. Table 6–3 on page 6-7 describes the types of system-created rule conditions that you can create with the `DBMS_STREAMS_ADM` package. If you need to create rules with complex conditions, then you can use the `DBMS_RULE_ADM` package.

There is a wide range of complex rule conditions. The following sections contain some examples of complex rule conditions.

Note:

- Complex rule conditions can degrade rule evaluation performance.
- In rule conditions, if you specify the name of a database, then make sure you include the full database name, including the domain name.

Rule Conditions Using the NOT Logical Condition to Exclude Objects You can use the NOT logical condition to exclude certain changes from being captured, propagated, applied, or dequeued in a Streams environment.

For example, suppose you want to specify rule conditions that evaluate to TRUE for all DML and DDL changes to all database objects in the hr schema, except for changes to the hr.regions table. You can use the NOT logical condition to accomplish this with two rules: one for DML changes and one for DDL changes. Here are the rule conditions for these rules:

```
(:dml.get_object_owner() = 'HR' AND NOT :dml.get_object_name() = 'REGIONS')
AND :dml.is_null_tag() = 'Y' ((:ddl.get_object_owner() = 'HR' OR :ddl.get_base_
table_owner() = 'HR') AND NOT :ddl.get_object_name() = 'REGIONS') AND :ddl.is_
null_tag() = 'Y'
```

Notice that object names, such as HR and REGIONS are specified in all uppercase characters in these examples. For rules to evaluate properly, the case of the characters in object names, such as tables and users, must match the case of the characters in the data dictionary. Therefore, if no case was specified for an object when the object was created, then specify the object name in all uppercase in rule conditions. However, if a particular case was specified through the use of double quotation marks when the objects was created, then specify the object name in the same case in rule conditions. However, the object name cannot be enclosed in double quotes in rule conditions.

For example, if the REGIONS table in the HR schema was actually created as "Regions", then specify Regions in rule conditions that involve this table, as in the following example:

```
:dml.get_object_name() = 'Regions'
```

You can use the Streams **evaluation context** when you create these rules using the DBMS_RULE_ADM package. The following example creates a rule set to hold the complex rules, creates rules with the previous conditions, and adds the rules to the rule set:

```
BEGIN
  -- Create the rule set
  DBMS_RULE_ADM.CREATE_RULE_SET(
    rule_set_name       => 'strmadmin.complex_rules',
    evaluation_context  => 'SYS.STREAMS$_EVALUATION_CONTEXT');
  -- Create the complex rules
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name  => 'strmadmin.hr_not_regions_dml',
    condition  => ' (:dml.get_object_owner() = ''HR'' AND NOT ' ||
                  ' :dml.get_object_name() = ''REGIONS'') AND ' ||
                  ' :dml.is_null_tag() = ''Y'' ');
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name  => 'strmadmin.hr_not_regions_ddl',
    condition  => ' ((:ddl.get_object_owner() = ''HR'' OR ' ||
                  ' :ddl.get_base_table_owner() = ''HR'') AND NOT ' ||
                  ' :ddl.get_object_name() = ''REGIONS'') AND ' ||
                  ' :ddl.is_null_tag() = ''Y'' ');
  --  Add the rules to the rule set
  DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'strmadmin.hr_not_regions_dml',
    rule_set_name  => 'strmadmin.complex_rules');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'strmadmin.hr_not_regions_ddl',
    rule_set_name  => 'strmadmin.complex_rules');
END;
```

```
/
```

In this case, the rules inherit the Streams evaluation context from the rule set.

Note: In most cases, you can avoid using complex rules with the NOT logical condition by using the DBMS_STREAMS_ADM package to add rules to the negative rule set for a Streams client

---

See Also: "System-Created Rules and Negative Rule Sets" on page 6-29

Rule Conditions Using the LIKE Condition You can use the LIKE condition to create complex rules that evaluate to TRUE when a condition in the rule matches a specified pattern. For example, suppose you want to specify rule conditions that evaluate to TRUE for all DML and DDL changes to all database objects in the hr schema that begin with the pattern JOB. You can use the LIKE condition to accomplish this with two rules: one for DML changes and one for DDL changes. Here are the rule conditions for these rules:

```
(:dml.get_object_owner() = 'HR' AND :dml.get_object_name() LIKE 'JOB%')
AND :dml.is_null_tag() = 'Y'

((:ddl.get_object_owner() = 'HR' OR :ddl.get_base_table_owner() = 'HR')
AND :ddl.get_object_name() LIKE 'JOB%') AND :ddl.is_null_tag() = 'Y'
```

Rule Conditions with Undefined Variables that Evaluate to NULL

During evaluation, an implicit variable in a rule condition is undefined if the variable value function for the variable returns NULL. An explicit variable without any attributes in a rule condition is undefined if the client does not send the value of the variable to the **rules engine** when it runs the DBMS_RULE.EVALUATE procedure.

Regarding variables with attributes, a variable is undefined if the client does not send the value of the variable, or any of its attributes, to the rules engine when it runs the DBMS_RULE.EVALUATE procedure. For example, if variable x has attributes a and b, then the variable is undefined if the client does not send the value of x and does not send the value of a and b. However, if the client sends the value of at least one attribute, then the variable is defined. In this case, if the client sends the value of a, but not b, then the variable is defined.

An undefined variable in a rule condition evaluates to NULL for Streams clients of the rules engine, which include capture processes, propagations, apply processes, and **messaging client**s. In contrast, for non-Streams clients of the rules engine, an undefined variable in a rule condition can cause the rules engine to return maybe_rules to the client. When a rule set is evaluated, maybe_rules are rules that might evaluate to TRUE given more information.

The number of maybe_rules returned to Streams clients is reduced by treating each undefined variable as NULL. Reducing the number of maybe_rules can improve performance if the reduction results in more efficient evaluation of a rule set when a message occurs. Rules that would result in maybe_rules for non-Streams clients can result in TRUE or FALSE rules for Streams clients, as the following examples illustrate.

Examples of Undefined Variables that Result in TRUE Rules for Streams Clients Consider the following user-defined rule condition:

```
:m IS NULL
```

If the value of the variable `m` is undefined during evaluation, then a maybe rule results for non-Streams clients of the rules engine. However, for Streams clients, this condition evaluates to `TRUE` because the undefined variable `m` is treated as a `NULL`. You should avoid adding rules such as this to rule sets for Streams clients, because such rules will evaluate to `TRUE` for every message. So, for example, if the positive rule set for a capture process has such a rule, then the capture process might capture messages that you did not intend to capture.

Here is another user-specified rule condition that uses a Streams `:dml` variable:

```
:dml.get_object_owner() = 'HR' AND :m IS NULL
```

For Streams clients, if a message consists of a row change to a table in the `hr` schema, and the value of the variable `m` is not known during evaluation, then this condition evaluates to `TRUE` because the undefined variable `m` is treated as a `NULL`.

Examples of Undefined Variables that Result in FALSE Rules for Streams Clients Consider the following user-defined rule condition:

```
:m = 5
```

If the value of the variable `m` is undefined during evaluation, then a maybe rule results for non-Streams clients of the rules engine. However, for Streams clients, this condition evaluates to `FALSE` because the undefined variable `m` is treated as a `NULL`.

Consider another user-specified rule condition that uses a Streams `:dml` variable:

```
:dml.get_object_owner() = 'HR' AND :m = 5
```

For Streams clients, if a message consists of a row change to a table in the `hr` schema, and the value of the variable `m` is not known during evaluation, then this condition evaluates to `FALSE` because the undefined variable `m` is treated as a `NULL`.

See Also: "Rule Set Evaluation" on page 5-10

Variables as Function Parameters in Rule Conditions

Oracle recommends that you avoid using `:dml` and `:ddl` variables as function parameters for rule conditions. The following example uses the `:dml` variable as a parameter to a function named `my_function`:

```
my_function(:dml) = 'Y'
```

Rule conditions such as these can degrade rule evaluation performance and can result in the capture or propagation of extraneous **Streams data dictionary** information.

See Also: "The Streams Data Dictionary" on page 2-37

User-Created Evaluation Contexts

You can use a custom **evaluation context** in a Streams environment. Any user-defined evaluation context involving LCRs must include all the variables in `SYS.STREAMS$_EVALUATION_CONTEXT`. The type of each variable and its variable value function must be the same for each variable as the ones defined in `SYS.STREAMS$_EVALUATION_CONTEXT`. In addition, when creating the evaluation context using `DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT`, the `SYS.DBMS_STREAMS_INTERNAL.EVALUATION_CONTEXT_FUNCTION` must be specified for the `evaluation_function` parameter. You can alter an existing evaluation context using the `DBMS_RULE_ADM.ALTER_EVALUATION_CONTEXT` procedure.

You can find information about an evaluation context in the following data dictionary views:

- `ALL_EVALUATION_CONTEXT_TABLES`
- `ALL_EVALUATION_CONTEXT_VARS`
- `ALL_EVALUATION_CONTEXTS`

If necessary, you can use the information in these data dictionary views to build a new evaluation context based on the `SYS.STREAMS$_EVALUATION_CONTEXT`.

Note: Avoid using variable names with special characters, such as $ and #, to ensure that there are no conflicts with Oracle-supplied evaluation context variables.

See Also: *Oracle Database Reference* for more information about these data dictionary views

7

Rule-Based Transformations

A **rule-based transformation** is any modification to a **message** when a **rule** in a **positive rule set** evaluates to `TRUE`. There are two types of rule-based transformations: declarative and custom. This chapter describes concepts related to rule-based transformations.

- Declarative Rule-Based Transformations
- Custom Rule-Based Transformations
- Rule-Based Transformations and Streams Clients
- Transformation Ordering
- Considerations for Rule-Based Transformations

See Also:

- Chapter 15, "Managing Rule-Based Transformations"
- Chapter 24, "Monitoring Rule-Based Transformations"

Declarative Rule-Based Transformations

Declarative rule-based transformations cover a set of common transformation scenarios for row LCRs. You specify (or declare) such a transformation using one of the following procedures in the `DBMS_STREAMS_ADM` package:

- `ADD_COLUMN` either adds or removes a declarative transformation that adds a column to a row LCR.
- `DELETE_COLUMN` either adds or removes a declarative transformation that deletes a column from a row LCR.
- `RENAME_COLUMN` either adds or removes a declarative transformation that renames a column in a row LCR.
- `RENAME_SCHEMA` either adds or removes a declarative transformation that renames the schema in a row LCR.
- `RENAME_TABLE` either adds or removes a declarative transformation that renames the table in a row LCR.

When you run one of these procedures to add a transformation, you specify the **rule** that is associated with the declarative rule-based transformation. When the specified rule evaluates to `TRUE` for a row LCR, Streams performs the declarative transformation internally on the row LCR, without invoking PL/SQL.

Declarative rule-based transformations provide the following advantages:

- Performance is improved because the transformations are run internally without using PL/SQL.
- Complexity is reduced because custom PL/SQL functions are not required.

Note:

- Declarative rule-based transformations can transform row LCRs only. These row LCRs can be captured row LCRs or user-enqueued row LCRs. Therefore, a DML rule must be specified when you run one of the procedures to add a declarative transformation. If a DDL rule is specified, then an error is raised.
- `ADD_COLUMN` transformations cannot add columns of the following datatypes: `BLOB`, `CLOB`, `NCLOB`, `BFILE`, `LONG`, `LONG RAW`, `ROWID`, and user-defined types (including object types, `REF`s, varrays, nested tables, and Oracle-supplied object types). The other declarative rule-based transformations that operate on columns support the same datatypes that are supported by Streams **capture process**es.

See Also:

- "Managing Declarative Rule-Based Transformations" on page 15-1
- "Row LCRs" on page 2-3
- "Datatypes Captured" on page 2-6 for more information about the datatypes supported by Streams capture processes

Custom Rule-Based Transformations

Custom rule-based transformations require a user-defined PL/SQL function to perform the transformation. The function takes as input an `ANYDATA` object containing a **message** and returns either an `ANYDATA` object containing the transformed message or an array that contains zero or more `ANYDATA` encapsulations of a message. A custom rule-based transformation function that returns one message is a one-to-one transformation function. A custom rule-based transformation function that can return more than one message in an array is a one-to-many transformation function. One-to-one transformation functions are supported for any type of **Streams client**, but one-to-many transformation functions are supported only for Streams **capture process**es.

To specify a custom rule-based transformation, use the `DBMS_STREAMS_ADM.SET_RULE_TRANSFORM_FUNCTION` procedure. You can use a custom rule-based transformation to modify both captured and **user-enqueued message**s, and these messages can be LCRs or **user message**s.

For example, a custom rule-based transformation can be used when the datatype of a particular column in a table is different at two different databases. The column might be a `NUMBER` column in the **source database** and a `VARCHAR2` column in the **destination database**. In this case, the transformation takes as input an `ANYDATA` object containing a row LCR with a `NUMBER` datatype for a column and returns an `ANYDATA` object containing a row LCR with a `VARCHAR2` datatype for the same column.

Other examples of custom transformations on messages include:

- Splitting a column into several columns
- Combining several columns into one column
- Modifying the contents of a column
- Modifying the payload of a user message

Custom rule-based transformations provide the following advantages:

- Flexibility is increased because you can use PL/SQL to perform custom transformations.
- A wider range of messages can be transformed, including DDL LCRs and user messages, as well as row LCRs.

The following considerations apply to custom rule-based transformations:

- When you perform custom rule-based transformations on DDL LCRs, you probably need to modify the DDL text in the DDL LCR to match any other modifications. For example, if the rule-based transformation changes the name of a table in the DDL LCR, then the rule-based transformation should change the table name in the DDL text in the same way.
- If possible, avoid specifying a custom rule-based transformation for a **global rule** or **schema rule** if the transformation pertains to a relatively small number of LCRs that will evaluate to `TRUE` for the **rule**. For example, a custom rule-based transformation that operates on a single table can be specified for a schema rule, and this schema can contain hundreds of tables. Specifying such a rule-based transformation has performance implications because extra processing is required for the LCRs that will not be transformed.

  To avoid specifying such a custom rule-based transformation, either you can use a **DML handler** to perform the transformation, or you can specify the transformation for a **table rule** instead of a global or schema rule. However, replacing a global or schema rule with table rules results in an increase in the total number of rules and additional maintenance when a new table is added.
- When a custom rule-based transformation that uses a one-to-one transformation function receives a **captured message**, the transformation can construct a new LCR and return it. Similarly, when a custom rule-based transformation that uses a one-to-many transformation function receives a captured message, the transformation can construct multiple new LCRs and return them in an array.

  For any LCR constructed and returned by a custom rule-based transformation, the `source_database_name`, `transaction_id`, and `scn` parameter values must match the values in the original LCR. Oracle automatically specifies the values in the original LCR for these parameters, even if an attempt is made to construct LCRs with different values.
- A custom rule-based transformation that receives a user-enqueued message can construct a new message and return it. In this case, the returned message can be an LCR constructed by the custom rule-based transformation.
- A custom rule-based transformation cannot convert an LCR into a non-LCR message. This restriction applies to captured messages and user-enqueued LCRs.
- A custom rule-based transformation cannot convert a row LCR into a DDL LCR or a DDL LCR into a row LCR. This restriction applies to captured messages and user-enqueued LCRs.

See Also:

- "How Rules Are Used in Streams" on page 6-1 for more information about global, schema, and table rules
- "Message Processing with an Apply Process" on page 4-2 for more information about DML handlers
- *Oracle Database PL/SQL Packages and Types Reference* for information about the SET_RULE_TRANSFORM_FUNCTION procedure

Custom Rule-Based Transformations and Action Contexts

You use the SET_RULE_TRANSFORM_FUNCTION procedure in the DBMS_STREAMS_ADM package to specify a custom rule-based transformation for a **rule**. This procedure modifies the **action context** of a rule to specify the transformation. A rule action context is optional information associated with a rule that is interpreted by the client of the **rules engine** after the rule evaluates to TRUE for a **message**. The client of the rules engine can be a user-created application or an internal feature of Oracle, such as Streams. The information in an action context is an object of type SYS.RE$NV_LIST, which consists of a list of name-value pairs.

A custom rule-based transformation in Streams always consists of the following name-value pair in an action context:

- If the function is a one-to-one transformation function, then the name is STREAMS$_TRANSFORM_FUNCTION. If the function is a one-to-many transformation function, then the name is STREAMS$_ARRAY_TRANS_FUNCTION.
- The value is an ANYDATA instance containing a PL/SQL function name specified as a VARCHAR2. This function performs the transformation.

You can display the existing custom rule-based transformations in a database by querying the DBA_STREAMS_TRANSFORM_FUNCTION data dictionary view.

When a rule in a **positive rule set** evaluates to TRUE for a message in a Streams environment, and an action context that contains a name-value pair with the name STREAMS$_TRANSFORM_FUNCTION or STREAMS$_ARRAY_TRANS_FUNCTION is returned, the PL/SQL function is run, taking the message as an input parameter. Other names in an action context beginning with STREAMS$_ are used internally by Oracle and must not be directly added, modified, or removed. Streams ignores any name-value pair that does not begin with STREAMS$_ or APPLY$_.

When a rule evaluates to FALSE for a message in a Streams environment, the rule is not returned to the client, and any PL/SQL function appearing in a name-value pair in the action context is not run. Different rules can use the same or different transformations. For example, different transformations can be associated with different operation types, tables, or schemas for which messages are being captured, propagated, applied, or dequeued.

Required Privileges for Custom Rule-Based Transformations

The user who calls the transformation function must have EXECUTE privilege on the function. The following list describes which user calls the transformation function:

- If a transformation is specified for a **rule** used by a **capture process**, then the **capture user** for the capture process calls the transformation function.
- If a transformation is specified for a rule used by a **propagation**, then the owner of the **source queue** for the propagation calls the transformation function.

- If a transformation is specified on a rule used by an **apply process**, then the **apply user** for the apply process calls the transformation function.
- If a transformation is specified on a rule used by a **messaging client**, then the user who invokes the messaging client calls the transformation function.

Rule-Based Transformations and Streams Clients

The following sections provide more information about rule-based transformations and **Streams client**s:

- Rule-Based Transformations and Capture Processes
- Rule-Based Transformations and Propagations
- Rule-Based Transformations and an Apply Process
- Rule-Based Transformations and a Messaging Client
- Multiple Rule-Based Transformations

The information in this section applies to both declarative and **custom rule-based transformation**s.

See Also:

- Chapter 15, "Managing Rule-Based Transformations"
- "Rule Action Context" on page 5-8
- "Message Processing with an Apply Process" on page 4-2 for more information about **DML handler**s

Rule-Based Transformations and Capture Processes

For a transformation to be performed during capture, a **rule** that is associated with a rule-based transformation in the **positive rule set** for the **capture process** must evaluate to `TRUE` for a particular change found in the redo log.

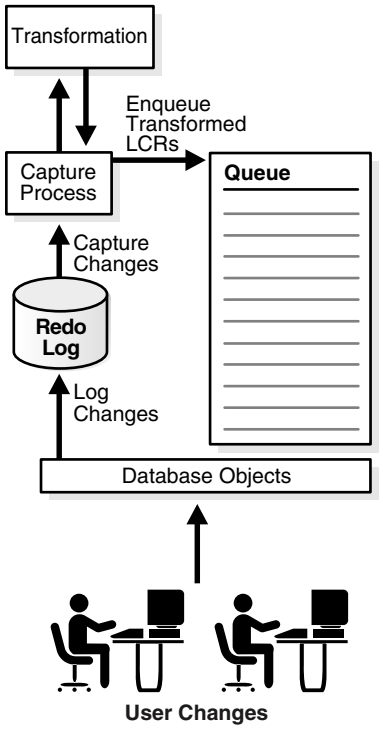
If the transformation is a **declarative rule-based transformation**, then Oracle transforms the **captured message** internally when the rule in a positive rule set evaluates to `TRUE` for the message. If the transformation is a **custom rule-based transformation**, then an **action context** containing a name-value pair with the name `STREAMS$_TRANSFORM_FUNCTION` or `STREAMS$_ARRAY_TRANS_FUNCTION` is returned to the capture process when the rule in a positive rule set evaluates to `TRUE` for the captured message.

The capture process completes the following steps to perform a rule-based transformation:

1. Formats the change in the redo log into an LCR.
2. Converts the LCR into an `ANYDATA` object.
3. Transforms the LCR. If the transformation is a declarative rule-based transformation, then Oracle transforms the `ANYDATA` object internally based on the specifications of the declarative transformation. If the transformation is a custom rule-based transformation, then the **capture user** runs the PL/SQL function in the name-value pair to transform the `ANYDATA` object.
4. Enqueues the one or more transformed `ANYDATA` objects into the **queue** associated with the capture process, or discards the LCR if an array that contains zero elements is returned by the transformation function.

All actions are performed by the capture user. Figure 7–1 shows a transformation during capture.

Figure 7–1 Transformation During Capture



For example, if an LCR is transformed during capture, then the transformed LCR is enqueued into the queue used by the capture process. Therefore, if such a captured message is propagated from the `dbs1.net` database to the `dbs2.net` and the `dbs3.net` databases, then the queues at `dbs2.net` and `dbs3.net` will contain the transformed LCR after propagation.

The advantages of performing transformations during capture are the following:

- Security can be improved if the transformation removes or changes private information, because this private information does not appear in the **source queue** and is not propagated to any **destination queue**.
- Space consumption can be reduced, depending on the type of transformation performed. For example, a transformation that reduces the amount of data results in less data to enqueue, propagate, and apply.
- Transformation overhead is reduced when there are multiple destinations for a transformed LCR, because the transformation is performed only once at the source, not at multiple destinations.
- A capture process transformation can transform a single message into multiple messages.

The possible disadvantages of performing transformations during capture are the following:

- The transformation overhead occurs in the **source database** if the capture process is a **local capture process**. However, if the capture process is a **downstream capture process**, then this overhead occurs at the **downstream database**, not at the source database.
- All sites receive the transformed LCR.

Attention: A rule-based transformation cannot be used with a capture process to modify or remove a column of a datatype that is not supported by Streams.

See Also: "Datatypes Captured" on page 2-6.

Rule-Based Transformation Errors During Capture

If an error occurs when the transformation function is run during capture, then the change is not captured, the error is returned to the capture process, and the capture process is disabled. Before the capture process can be enabled, you must either change or remove the rule-based transformation to avoid the error.

Rule-Based Transformations and Propagations

For a transformation to be performed during propagation, a **rule** that is associated with a rule-based transformation in the **positive rule set** for the **propagation** must evaluate to `TRUE` for a **message** in the **source queue** for the propagation. This message can be a **captured message** or a **user-enqueued message**.

If the transformation is a **declarative rule-based transformation**, then Oracle transforms the message internally when the rule in a positive rule set evaluates to `TRUE` for the message. If the transformation is a **custom rule-based transformation**, then an **action context** containing a name-value pair with the name `STREAMS$_TRANSFORM_FUNCTION` is returned to the propagation when the rule in a positive rule set evaluates to `TRUE` for the message.

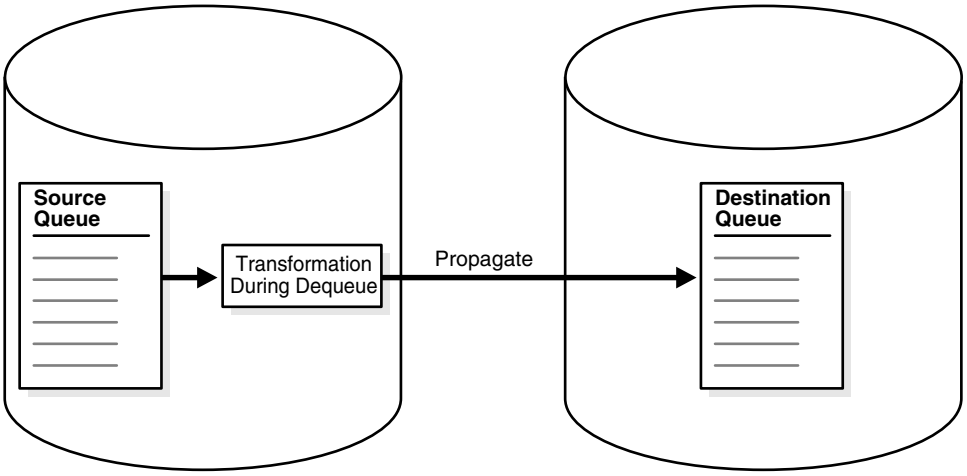
The propagation completes the following steps to perform a rule-based transformation:

1. Starts dequeuing the message from the source queue.
2. Transforms the message. If the transformation is a declarative rule-based transformation, then Oracle transforms the message internally based on the specifications of the declarative transformation. If the transformation is a custom rule-based transformation, then the source queue owner runs the PL/SQL function in the name-value pair to transform the message.
3. Completes dequeuing the transformed message.
4. Propagates the transformed message to the **destination queue**.

See Also: "Captured and User-Enqueued Messages in an ANYDATA Queue" on page 3-3

Figure 7–2 shows a transformation during propagation.

Figure 7–2 Transformation During Propagation



For example, suppose you use a rule-based transformation for a propagation that propagates messages from the `dbs1.net` database to the `dbs2.net` database, but you do not use a rule-based transformation for a propagation that propagates messages from the `dbs1.net` database to the `dbs3.net` database.

In this case, a message in the queue at `dbs1.net` can be transformed before it is propagated to `dbs2.net`, but the same message can remain in its original form when it is propagated to `dbs3.net`. In this case, after propagation, the queue at `dbs2.net` contains the transformed message, and the queue at `dbs3.net` contains the original message.

The advantages of performing transformations during propagation are the following:

- Security can be improved if the transformation removes or changes private information before messages are propagated.
- Some destination queues can receive a transformed message, while other destination queues can receive the original message.
- Different destinations can receive different variations of the same transformed message.

The possible disadvantages of performing transformations during propagation are the following:

- Once a message is transformed, any database to which it is propagated after the first propagation receives the transformed message. For example, if `dbs2.net` propagates the message to `dbs4.net`, then `dbs4.net` receives the transformed message.
- When the first propagation in a **directed network** performs the transformation, and the **capture process** that captured the message is local, the transformation overhead occurs on the **source database**. However, if the capture process is a **downstream capture process**, then this overhead occurs at the **downstream database**, not at the source database.
- The same transformation can be done multiple times on a message when different propagations send the message to multiple **destination database**s.

Rule-Based Transformation Errors During Propagation

If an error occurs during the transformation, then the message that caused the error is not dequeued or propagated, and the error is returned to the propagation. Before the message can be propagated, you must change or remove the rule-based transformation to avoid the error.

Rule-Based Transformations and an Apply Process

For a transformation to be performed during apply, a **rule** that is associated with a rule-based transformation in the **positive rule set** for the **apply process** must evaluate to `TRUE` for a **message** in the **queue** for the apply process. This message can be a **captured message** or a **user-enqueued message**.

If the transformation is a **declarative rule-based transformation**, then Oracle transforms the message internally when the rule in a positive rule set evaluates to `TRUE` for the message. If the transformation is a **custom rule-based transformation**, then an **action context** containing a name-value pair with the name `STREAMS$_TRANSFORM_FUNCTION` is returned to the apply process when the rule in a positive rule set evaluates to `TRUE` for the message.

The apply process completes the following steps to perform a rule-based transformation:

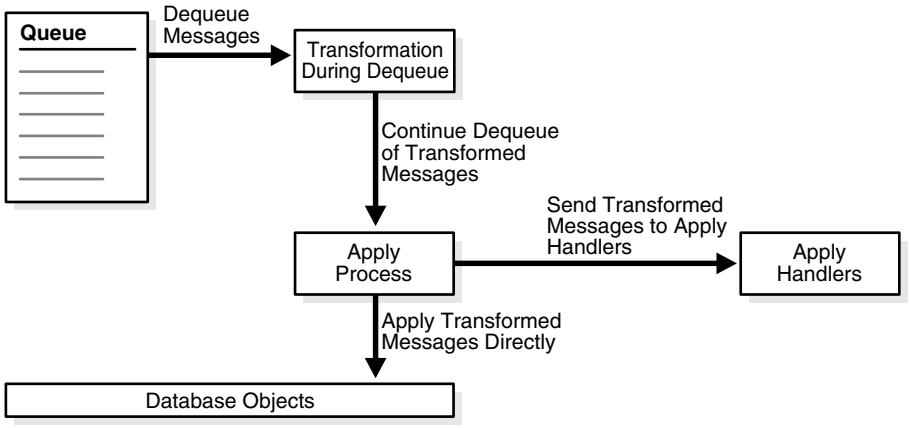
1. Starts to dequeue the message from the queue.
2. Transforms the message. If the transformation is a declarative rule-based transformation, then Oracle transforms the message internally based on the specifications of the declarative transformation. If the transformation is a custom rule-based transformation, then the **apply user** runs the PL/SQL function in the name-value pair to transform the message.
3. Completes dequeuing the transformed message.
4. Applies the transformed message, which can entail changing database objects at the **destination database** or sending the transformed message to an **apply handler**.

All actions are performed by the apply user.

> **See Also:** "Captured and User-Enqueued Messages in an ANYDATA Queue" on page 3-3

Figure 7–3 shows a transformation during apply.

Figure 7–3 Transformation During Apply



For example, suppose a message is propagated from the `dbs1.net` database to the `dbs2.net` database in its original form. When the apply process dequeues the message from a queue at `dbs2.net`, the message is transformed.

The possible advantages of performing transformations during apply are the following:

- Any database to which the message is propagated after the first propagation can receive the message in its original form. For example, if `dbs2.net` propagates the message to `dbs4.net`, then `dbs4.net` can receive the original message.
- The transformation overhead does not occur on the **source database** when the source and destination database are different.

The possible disadvantages of performing transformations during apply are the following:

- Security might be a concern if the messages contain private information, because all databases to which the messages are propagated receive the original messages.
- The same transformation can be done multiple times when multiple destination databases need the same transformation.

> **Note:** Before modifying one or more rules for an apply process, you should stop the apply process.

Rule-Based Transformation Errors During Apply Process Dequeue

If an error occurs when the transformation function is run during apply process dequeue, then the message that caused the error is not dequeued, the transaction containing the message is not applied, the error is returned to the apply process, and the apply process is disabled. Before the apply process can be enabled, you must change or remove the rule-based transformation to avoid the error.

Apply Errors on Transformed Messages

If an apply error occurs for a transaction in which some of the messages have been transformed by a rule-based transformation, then the transformed messages are moved to the error queue with all of the other messages in the transaction. If you use the `EXECUTE_ERROR` procedure in the `DBMS_APPLY_ADM` package to reexecute a transaction in the error queue that contains transformed messages, then the

transformation is not performed on the messages again because the apply process rule set containing the rule is not evaluated again.

Rule-Based Transformations and a Messaging Client

For a transformation to be performed during dequeue by a **messaging client**, a **rule** that is associated with a rule-based transformation in the **positive rule set** for the messaging client must evaluate to `TRUE` for a **message** in the **queue** for the messaging client.

If the transformation is a **declarative rule-based transformation**, then Oracle transforms the message internally when the rule in a positive rule set evaluates to `TRUE` for the message. If the transformation is a **custom rule-based transformation**, then an **action context** containing a name-value pair with the name `STREAMS$_TRANSFORM_FUNCTION` is returned to the messaging client when the rule in a positive rule set evaluates to `TRUE` for the message.

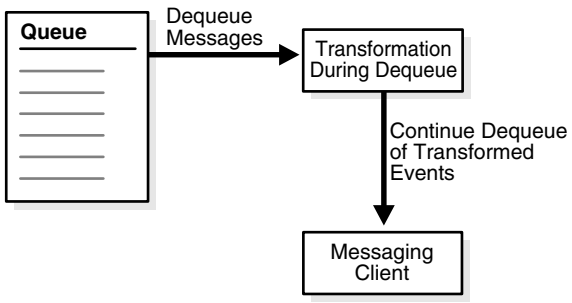
The messaging client completes the following steps to perform a rule-based transformation:

1. Starts to dequeue the message from the queue
2. Transforms the message. If the transformation is a declarative rule-based transformation, then the message must be a user-enqueued row LCR, and Oracle transforms the row LCR internally based on the specifications of the declarative transformation. If the transformation is a **custom rule-based transformation**, then the message can be a user-enqueued row LCR, DDL LCR, or message, and the user who invokes the messaging client runs the PL/SQL function in the name-value pair to transform the message during dequeue.
3. Completes dequeuing the transformed message

All actions are performed by the user who invokes the messaging client.

Figure 7–4 shows a transformation during messaging client dequeue.

Figure 7–4 Transformation During Messaging Client Dequeue



For example, suppose a message is propagated from the `dbs1.net` database to the `dbs2.net` database in its original form. When the messaging client dequeues the message from a queue at `dbs2.net`, the message is transformed.

One possible advantage of performing transformations during dequeue in a messaging environment is that any database to which the message is propagated after the first propagation can receive the message in its original form. For example, if `dbs2.net` propagates the message to `dbs4.net`, then `dbs4.net` can receive the original message.

The possible disadvantages of performing transformations during dequeue in a messaging environment are the following:

- Security might be a concern if the messages contain private information, because all databases to which the messages are propagated receive the original messages.
- The same transformation can be done multiple times when multiple **destination database**s need the same transformation.

Rule-Based Transformation Errors During Messaging Client Dequeue

If an error occurs when the transformation function is run during messaging client dequeue, then the message that caused the error is not dequeued, and the error is returned to the messaging client. Before the message can be dequeued by the messaging client, you must change or remove the rule-based transformation to avoid the error.

Multiple Rule-Based Transformations

You can transform a message during capture, propagation, apply, or dequeue, or during any combination of capture, propagation, apply, and dequeue. For example, if you want to hide sensitive data from all recipients, then you can transform a message during capture. If some recipients require additional custom transformations, then you can transform the previously transformed message during propagation, apply, or dequeue.

Transformation Ordering

In addition to **declarative rule-based transformation**s and **custom rule-based transformation**s, a **row migration** is an internal transformation that takes place when a **subset rule** evaluates to `TRUE`. If all three types of transformations are specified for a single **rule**, then Oracle performs the transformations in the following order when the rule evaluates to `TRUE`:

1. Row migration
2. Declarative rule-based transformation
3. Custom rule-based transformation

Declarative Rule-Based Transformation Ordering

If more than one **declarative rule-based transformation** is specified for a single **rule**, then Oracle must perform the transformations in a particular order. You can use the default ordering for declarative transformations, or you can specify the order.

Default Declarative Transformation Ordering

By default, Oracle performs declarative transformations in the following order when the rule evaluates to `TRUE`:

1. Delete column
2. Rename column
3. Add column
4. Rename table
5. Rename schema

The results of a declarative transformation are used in each subsequent declarative transformation. For example, suppose the following declarative transformations are specified for a single rule:

- Delete column `address`
- Add column `address`

Assuming column `address` exists in a row LCR, both declarative transformations should be performed in this case because column `address` is deleted from the row LCR before column `address` is added back to the row LCR. The following table shows the transformation ordering for this example.

Step Number	Transformation Type	Transformation Details	Transformation Performed?
1	Delete column	Delete column `address` from row LCR	Yes
2	Rename column	-	-
3	Add column	Add column `address` to row LCR	Yes
4	Rename table	-	-
5	Rename schema	-	-

Another scenario might rename a table and then rename a schema. For example, suppose the following declarative transformations are specified for a single rule:

- Rename table `john.customers` to `sue.clients`
- Rename schema `sue` to `mary`

Notice that the rename table transformation also renames the schema for the table. In this case, both transformations should be performed and, after both transformations, the table name becomes `mary.clients`. The following table shows the transformation ordering for this example.

Step Number	Transformation Type	Transformation Details	Transformation Performed?
1	Delete column	-	-
2	Rename column	-	-
3	Add column	-	-
4	Rename table	Rename table `john.customers` to `sue.clients`	Yes
5	Rename schema	Rename schema `sue` to `mary`	Yes

Consider a similar scenario in which the following declarative transformations are specified for a single rule:

- Rename table `john.customers` to `sue.clients`
- Rename schema `john` to `mary`

In this case, the first transformation is performed, but the second one is not. After the first transformation, the table name is `sue.clients`. The second transformation is not performed because the schema of the table is now `sue`, not `john`. The following table shows the transformation ordering for this example.

Step Number	Transformation Type	Transformation Details	Transformation Performed?
1	Delete column	-	-
2	Rename column	-	-
3	Add column	-	-
4	Rename table	Rename table `john.customers` to `sue.clients`	Yes
5	Rename schema	Rename schema `john` to `mary`	No

The rename schema transformation is not performed, but it does not result in an error. In this case, the row LCR is transformed by the rename table transformation, and a row LCR with the table name `sue.clients` is returned.

User-Specified Declarative Transformation Ordering

If you do not want to use the default declarative rule-based transformation ordering for a particular rule, then you can specify step numbers for each declarative transformation specified for the rule. If you specify a step number for one or more declarative transformations for a particular rule, then the declarative transformations for the rule behave in the following way:

- Declarative transformations are performed in order of increasing step number.
- The default step number for a declarative transformation is `0` (zero). A declarative transformation uses this default if no step number is specified for it explicitly.
- If two or more declarative transformations have the same step number, then these declarative transformations follow the default ordering described in "Default Declarative Transformation Ordering" on page 7-12.

For example, you can reverse the default ordering for declarative transformations by specifying the following step numbers for transformations associated with a particular rule:

- Delete column with step number 5
- Rename column with step number 4
- Add column with step number 3
- Rename table with step number 2
- Rename schema with step number 1

With this ordering specified, rename schema transformations are performed first, and delete column transformations are performed last.

Considerations for Rule-Based Transformations

The following considerations apply to both **declarative rule-based transformation**s and **custom rule-based transformation**s:

- For a rule-based transformation to be performed by a **Streams client**, the **rule** must be in the **positive rule set** for the Streams client. If the rule is in the **negative rule set** for the Streams client, then the Streams client ignores the rule-based transformation.
- Rule-based transformations are different from transformations performed using the `DBMS_TRANSFORM` package. This document does not discuss transformations performed with the `DBMS_TRANSFORM` package.
- If a large percentage of row LCRs will be transformed in your environment, or if you need to make expensive transformations on row LCRs, then consider making these modifications within a **DML handler** instead, because DML handlers can execute in parallel when apply parallelism is greater than 1.

> **See Also:** *Oracle Streams Advanced Queuing User's Guide and Reference* and *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_TRANSFORM` package

8

Information Provisioning

Information provisioning makes information available when and where it is needed. Information provisioning is part of Oracle grid computing, which pools large numbers of servers, storage areas, and networks into a flexible, on-demand computing resource for enterprise computing needs. Information provisioning uses many of the features that also are used for information integration.

This chapter contains these topics:

- Overview of Information Provisioning
- Bulk Provisioning of Large Amounts of Information
- Incremental Information Provisioning with Streams
- On-Demand Information Access

See Also:

- Chapter 16, "Using Information Provisioning"
- *Oracle Database Concepts* for more information about information integration

Overview of Information Provisioning

Oracle grid computing enables resource provisioning with features such as Oracle Real Application Clusters (RAC), Oracle Scheduler, and Database Resource Manager. RAC enables you to provision hardware resources by running a single Oracle database server on a cluster of physical servers. Oracle Scheduler enables you to provision database workload over time for more efficient use of resources. Database Resource Manager provisions resources to database users, applications, or services within an Oracle database.

In addition to resource provisioning, Oracle grid computing also enables information provisioning. Information provisioning delivers information when and where it is needed, regardless of where the information currently resides on the grid. In a grid environment with distributed systems, the grid must move or copy information efficiently to make it available where it is needed.

Information provisioning can take the following forms:

- Bulk Provisioning of Large Amounts of Information: Data Pump export/import, transportable tablespaces, the DBMS_STREAMS_TABLESPACE_ADM package, and the DBMS_FILE_TRANSFER package all are ways to provide large amounts of information. Data Pump export/import enables you to move or copy information at the database, tablespace, schema, or table level. Transportable tablespaces enables you to move or copy tablespaces from one database to another efficiently. The procedures in the DBMS_STREAMS_TABLESPACE_ADM package enable you to clone, detach, and attach tablespaces. In addition, some procedures in this package enable you to store tablespaces in a **tablespace repository** that provides versioning of tablespaces. When tablespaces are needed, they can be pulled from the tablespace repository and plugged into a database. The procedures in the DBMS_FILE_TRANSFER package enable you to copy a binary file within a database or between databases.
- Incremental Information Provisioning with Streams: Some data must be shared as it is created or changed, rather than occasionally shared in bulk. Oracle Streams can stream data between databases, nodes, or blade farms in a grid and can keep two or more copies synchronized as updates are made.
- On-Demand Information Access: You can make information available without moving or copying it to a new location. Oracle Distributed SQL allows grid users to access and integrate data stored in multiple Oracle databases and, through Gateways, non-Oracle databases.

These information provisioning capabilities can be used individually or in combination to provide a full information provisioning solution in your environment. The remaining sections in this chapter discuss the ways to provision information in more detail.

See Also:

- *Oracle Database Oracle Clusterware and Oracle Real Application Clusters Administration and Deployment Guide* for more information about RAC
- *Oracle Database Administrator's Guide* for information about Oracle Scheduler and Database Resource Manager

Bulk Provisioning of Large Amounts of Information

Oracle provides several ways to move or copy large amounts of information from database to database efficiently. Data Pump can export and import at the database, tablespace, schema, or table level. There are several ways to move or copy a tablespace set from one Oracle database to another. Transportable tablespaces can move or copy a subset of an Oracle database and "plug" it in to another Oracle database. Transportable tablespace from backup with RMAN enables you to move or copy a tablespace set while the tablespaces remain online. The procedures in the DBMS_STREAMS_TABLESPACE_ADM package combine several steps that are required to move or copy a tablespace set into one procedure call.

Each method for moving or copying a tablespace set requires that the tablespace set is self-contained. A self-contained tablespace has no references from the tablespace pointing outside of the tablespace. For example, if an index in the tablespace is for a table in a different tablespace, then the tablespace is not self-contained. A self-contained tablespace set has no references from inside the set of tablespaces pointing outside of the set of tablespaces. For example, if a partitioned table is partially contained in the set of tablespaces, then the set of tablespaces is not self-contained. To

determine whether a set of tablespaces is self-contained, use the TRANSPORT_SET_CHECK procedure in the Oracle supplied package DBMS_TTS.

The following sections describe the options for moving or copying large amounts of information and when to use each option:

- Data Pump Export/Import
- Transportable Tablespace from Backup with RMAN
- DBMS_STREAMS_TABLESPACE_ADM Procedures
- Options for Bulk Information Provisioning

Data Pump Export/Import

Data Pump export/import can move or copy data efficiently between databases. Data Pump can export/import a full database, tablespaces, schemas, or tables to provision large or small amounts of data for a particular requirement. Data Pump exports and imports can be performed using command line clients (expdp and impdp) or the DBMS_DATAPUMP package.

A transportable tablespaces export/import is specified using the TRANSPORT_TABLESPACES parameter. Transportable tablespaces enables you to unplug a set of tablespaces from a database, move or copy them to another location, and then plug them into another database. The transport is quick because the process transfers metadata and files. It does not unload and load the data. In transportable tablespaces mode, only the metadata for the tables (and their dependent objects) within a specified set of tablespaces are unloaded at the source and loaded at the target. This allows the tablespace datafiles to be copied to the target Oracle database and incorporated efficiently.

The tablespaces being transported can be either dictionary managed or locally managed. Moving or copying tablespaces using transportable tablespaces is faster than performing either an export/import or unload/load of the same data. To use transportable tablespaces, you must have the EXP_FULL_DATABASE and IMP_FULL_DATABASE role. The tablespaces being transported must be read-only during export, and the export cannot have a degree of parallelism greater than 1.

See Also:

- *Oracle Database Utilities* for more information about Data Pump
- *Oracle Database Administrator's Guide* for more information about using Data Pump with the TRANSPORT_TABLESPACES option

Transportable Tablespace from Backup with RMAN

The Recovery Manager (RMAN) TRANSPORT TABLESPACE command copies tablespaces without requiring that the tablespaces be in read-only mode during the transport process. Appropriate database backups must be available to perform RMAN transportable tablespace from backup.

See Also:

- *Oracle Database Backup and Recovery Reference*
- *Oracle Database Backup and Recovery Advanced User's Guide*

DBMS_STREAMS_TABLESPACE_ADM Procedures

The following procedures in the DBMS_STREAMS_TABLESPACE_ADM package can be used to move or copy tablespaces:

- ATTACH_TABLESPACES: Uses Data Pump to import a self-contained tablespace set previously exported using the DBMS_STREAMS_TABLESPACE_ADM package, Data Pump export, or the RMAN TRANSPORT TABLESPACE command.
- CLONE_TABLESPACES: Uses Data Pump export to clone a set of self-contained tablespaces. The tablespace set can be attached to a database after it is cloned. The tablespace set remains in the database from which it was cloned.
- DETACH_TABLESPACES: Uses Data Pump export to detach a set of self-contained tablespaces. The tablespace set can be attached to a database after it is detached. The tablespace set is dropped from the database from which it was detached.
- PULL_TABLESPACES: Uses Data Pump export/import to copy a set of self-contained tablespaces from a remote database and attach the tablespace set to the current database.

In addition, the DBMS_STREAMS_TABLESPACE_ADM package also contains the following procedures: ATTACH_SIMPLE_TABLESPACE, CLONE_SIMPLE_TABLESPACE, DETACH_SIMPLE_TABLESPACE, and PULL_SIMPLE_TABLESPACE. These procedures operate on a single tablespace that uses only one datafile instead of a tablespace set.

File Group Repository

In the context of a file group, a **file** is a reference to a file stored on hard disk. A file is composed of a file name, a directory object, and a file type. The directory object references the directory in which the file is stored on hard disk. A **version** is a collection of related files, and a **file group** is a collection of versions.

A **file group repository** is a collection of all of the file groups in a database. A file group repository can contain multiple file groups and multiple versions of a particular file group.

For example, a file group named reports can store versions of sales reports. The reports can be generated on a regular schedule, and each version can contain the report files. The file group repository can version the file group under names such as sales_reports_v1, sales_reports_v2, and so on.

File group repositories can contain all types of files. You can create and manage file group repositories using the DBMS_FILE_GROUP package.

See Also:

- "Using a File Group Repository" on page 16-14
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the DBMS_FILE_GROUP package

Tablespace Repository

A **tablespace repository** is a collection of tablespace sets in a file group repository. Tablespace repositories are built on file group repositories, but tablespace repositories only contain the files required to move or copy tablespaces between databases. A file group repository can store versioned sets of files, including, but not restricted to, tablespace sets.

Different tablespace sets can be stored in a tablespace repository, and different versions of a particular tablespace set can also be stored. A version of a tablespace set in a tablespace repository consists of the following files:

- The Data Pump export dump file for the tablespace set
- The Data Pump log file for the export
- The datafiles that make up the tablespace set

All of the files in a version can reside in a single directory, or they can reside in different directories. The following procedures can move or copy tablespaces with or without using a tablespace repository:

- `ATTACH_TABLESPACES`
- `CLONE_TABLESPACES`
- `DETACH_TABLESPACES`

If one of these procedures is run without using a tablespace repository, then a tablespace set is moved or copied, but it is not placed in or copied from a tablespace repository. If the `CLONE_TABLESPACES` or `DETACH_TABLESPACES` procedure is run using a tablespace repository, then the procedure places a tablespace set in the repository as a version of the tablespace set. If the `ATTACH_TABLESPACES` procedure is run using a tablespace repository, then the procedure copies a particular version of a tablespace set from the repository and attaches it to a database.

When to Use a Tablespace Repository A tablespace repository is useful when you need to store different versions of one or more tablespace sets. For example, a tablespace repository can be used to accomplish the following goals:

- You want to run quarterly reports on a tablespace set. You can clone the tablespace set quarterly for storage in a versioned tablespace repository, and a specific version of the tablespace set can be requested from the repository and attached to another database to run the reports.
- You want applications to be able to attach required tablespace sets on demand in a grid environment. You can store multiple versions of several different tablespace sets in the tablespace repository. Each tablespace set can be used for a different purpose by the application. When the application needs a particular version of a particular tablespace set, the application can scan the tablespace repository and attach the correct tablespace set to a database.

Differences Between the Tablespace Repository Procedures The procedures that include the `file_group_name` parameter in the `DBMS_STREAMS_TABLESPACE_ADM` package behave differently with regard to the tablespace set, the datafiles in the tablespace set, and the export dump file. Table 8–1 describes these differences.

Table 8–1 Tablespace Repository Procedures

Procedure	Tablespace Set	Datafiles	Export Dump File
ATTACH_TABLESPACES	The tablespace set is added to the local database.	If the datafiles_directory_object parameter is non-NULL, then the datafiles are copied from their current location(s) for the version in the tablespace repository to the directory object specified in the datafiles_directory_object parameter. The attached tablespace set uses the datafiles that were copied. If the datafiles_directory_object parameter is NULL, then the datafiles are not moved or copied. The datafiles remain in the directory object(s) for the version in the tablespace repository, and the attached tablespace set uses these datafiles.	If the datafiles_directory_object parameter is non-NULL, then the export dump file is copied from its directory object for the version in the tablespace repository to the directory object specified in the datafiles_directory_object parameter. If the datafiles_directory_object parameter is NULL, then the export dump file is not moved or copied.
CLONE_TABLESPACES	The tablespace set is retained in the local database.	The datafiles are copied from their current location(s) to the directory object specified in the tablespace_directory_object parameter or in the default directory for the version or **file group**. This parameter specifies where the version of the tablespace set is stored in the tablespace repository. The current location of the datafiles can be determined by querying the DBA_DATA_FILES data dictionary view. A directory object must exist, and must be accessible to the user who runs the procedure, for each datafile location.	The export dump file is placed in the directory object specified in the tablespace_directory_object parameter or in the default directory for the version or file group.
DETACH_TABLESPACES	The tablespace set is dropped from the local database.	The datafiles are not moved or copied. The datafiles remain in their current location(s). A directory object must exist, and must be accessible to the user who runs the procedure, for each datafile location. These datafiles are included in the version of the tablespace set stored in the tablespace repository.	The export dump file is placed in the directory object specified in the export_directory_object parameter or in the default directory for the version or file group.

Remote Access to a Tablespace Repository A tablespace repository can reside in the database that uses the tablespaces, or it can reside in a remote database. If it resides in a remote database, then a database link must be specified in the repository_db_link parameter when you run one of the procedures, and the database link must be accessible to the user who runs the procedure.

Only One Tablespace Version Can Be Online in a Database A version of a tablespace set in a tablespace repository can be either online or offline in a database. A tablespace set version is online in a database when it is attached to the database using the ATTACH_TABLESPACES procedure. Only a single version of a tablespace set can be online in a database at a particular time. However, the same version or different versions of a tablespace set can be online in different databases at the same time. In this case, it might be necessary to ensure that only one database can make changes to the tablespace set.

Tablespace Repository Procedures Use the DBMS_FILE_GROUP Package Automatically Although tablespace repositories are built on file group repositories, it is not necessary to use the DBMS_FILE_GROUP package to create a file group repository before using one of the procedures in the DBMS_STREAMS_TABLESPACE_ADM package. If you run the CLONE_TABLESPACES or DETACH_TABLESPACES procedure and specify a file group that does not exist, then the procedure creates the file group automatically.

A Tablespace Repository Provides Versioning but Not Source Control A tablespace repository provides versioning of tablespace sets, but it does not provide source control. If two or more versions of a tablespace set are changed at the same time and placed in a tablespace repository, then these changes are not merged.

Read-Only Tablespaces Requirement During Export

The procedures in the DBMS_STREAMS_TABLESPACE_ADM package that perform a Data Pump export make any read/write tablespace being exported read-only. After the export is complete, if a procedure in the DBMS_STREAMS_TABLESPACE_ADM package made a tablespace read-only, then the procedure makes the tablespace read/write.

Automatic Platform Conversion for Tablespaces

When one of the procedures in the DBMS_STREAMS_TABLESPACE_ADM package moves or copies tablespaces to a database that is running on a different platform, the procedure can convert the datafiles to the appropriate platform if the conversion is supported. The V$TRANSPORTABLE_PLATFORM dynamic performance view lists all platforms that support cross-platform transportable tablespaces.

When a tablespace repository is used, the platform conversion is automatic if it is supported. When a tablespace repository is not used, you must specify the platform to which or from which the tablespace is being converted.

See Also:

- Chapter 16, "Using Information Provisioning" for information about using the procedures in the DBMS_STREAMS_TABLESPACE_ADM package, including usage scenarios
- *Oracle Database PL/SQL Packages and Types Reference* for reference information about the DBMS_STREAMS_TABLESPACE_ADM package and the DBMS_FILE_GROUP package

Options for Bulk Information Provisioning

Table 8–2 describes when to use each option for bulk information provisioning.

Table 8–2 Options for Moving or Copying Tablespaces

Option	Use this Option Under these Conditions
Data Pump export/import	■ You want to move or copy data at the database, tablespace, schema, or table level. ■ You want to perform each step required to complete the Data Pump export/import.
Data Pump export/import with the `TRANSPORT_TABLESPACES` option	■ The tablespaces being moved or copied can be read-only during the operation. ■ You want to perform each step required to complete the Data Pump export/import.
Transportable tablespace from backup with the RMAN `TRANSPORT TABLESPACE` command	The tablespaces being moved or copied must remain online (writeable) during the operation.
`DBMS_STREAMS_TABLESPACE_ADM` procedures without a tablespace repository	■ The tablespaces being moved or copied can be read-only during the operation. ■ You want to combine multiple steps in the Data Pump export/import into one procedure call. ■ You do not want to use a tablespace repository for the tablespaces being moved or copied.
`DBMS_STREAMS_TABLESPACE_ADM` procedures with a tablespace repository	■ The tablespaces being moved or copied can be read-only during the operation. ■ You want to combine multiple steps in the Data Pump export/import into one procedure call. ■ You want to use a tablespace repository for the tablespaces being moved or copied. ■ You want platform conversion to be automatic.

Incremental Information Provisioning with Streams

Streams can share and maintain database objects in different databases at each of the following levels:

- Database
- Schema
- Table
- Table subset

Streams can keep shared database objects synchronized at two or more databases. Specifically, a Streams **capture process** captures changes to a shared database object in a **source database**'s redo log, one or more **propagation**s propagate the changes to another database, and a Streams **apply process** applies the changes to the shared database object. If database objects are not identical at different databases, then Streams can transform them at any point in the process. That is, a change can be transformed during capture, propagation, or apply. In addition, Streams provides custom processing of changes during apply with apply handlers. Database objects can be shared between Oracle databases, or they can be shared between Oracle and non-Oracle databases through the use of Oracle Transparent Gateways. In addition to data **replication**, Streams provides messaging, event management and notification, and data warehouse loading.

A combination of Streams and bulk provisioning enables you to copy and maintain a large amount of data by running a single procedure. The following procedures in the DBMS_STREAMS_ADM package use Data Pump to copy data between databases and configure Streams to maintain the copied data incrementally:

- MAINTAIN_GLOBAL configures a Streams environment that replicates changes at the database level between two databases.
- MAINTAIN_SCHEMAS configures a Streams environment that replicates changes to specified schemas between two databases.
- MAINTAIN_SIMPLE_TTS clones a simple tablespace from a **source database** to a **destination database** and uses Streams to maintain this tablespace at both databases.
- MAINTAIN_TABLES configures a Streams environment that replicates changes to specified tables between two databases.
- MAINTAIN_TTS uses transportable tablespaces with Data Pump to clone a set of tablespaces from a source database to a destination database and uses Streams to maintain these tablespaces at both databases.

In addition, the PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP procedures configure a Streams environment that replicates changes either at the database level or to specified tablespaces between two databases. These procedures must be used together, and **instantiation** actions must be performed manually, to complete the Streams replication configuration.

Using these procedures, you can export data from one database, ship it to another database, reformat the data if the second database is on a different platform, import the data into the second database, and start syncing the data with the changes happening in the first database. If the second database is on a grid, then you have just migrated your application to a grid with one command.

These procedures can configure **Streams client**s to maintain changes originating at the source database in a single-source replication environment, or they can configure Streams clients to maintain changes originating at both databases in a bidirectional replication environment. By maintaining changes to the data, it can be kept synchronized at both databases. These procedures can either perform these actions directly, or they can generate one or more scripts that performs these actions.

See Also:

- Chapter 1, "Introduction to Streams"
- *Oracle Database PL/SQL Packages and Types Reference* for reference information about the DBMS_STREAMS_ADM package
- *Oracle Streams Replication Administrator's Guide* for information about using the DBMS_STREAMS_ADM package

On-Demand Information Access

Users and applications can access information without moving or copying it to a new location. Distributed SQL allows grid users to access and integrate data stored in multiple Oracle and, through Oracle Transparent Gateways, non-Oracle databases. Transparent remote data access with distributed SQL allows grid users to run their applications against any other database without making any code change to the applications. While integrating data and managing transactions across multiple data stores, the Oracle database optimizes the execution plans to access data in the most efficient manner.

See Also:

- *Oracle Database Administrator's Guide* for information about distributed SQL
- *Oracle Database Heterogeneous Connectivity Administrator's Guide* for more information about Oracle Transparent Gateways

9

Streams High Availability Environments

This chapter explains concepts relating to Streams high availability environments.

This chapter contains these topics:

Overview of Streams High Availability Environments

Configuring a high availability solution requires careful planning and analysis of failure scenarios. Database backups and physical standby databases provide physical copies of a source database for failover protection. Oracle Data Guard, in SQL apply mode, implements a logical standby database in a high availability environment. Because Oracle Data Guard is designed for a high availability environment, it handles most failure scenarios. However, some environments might require the flexibility available in Oracle Streams, so that they can take advantage of the extended feature set offered by Streams.

This chapter discusses some of the scenarios that can benefit from a Streams-based solution and explains Streams-specific issues that arise in high availability environments. It also contains information about best practices for deploying Streams in a high availability environment, including hardware failover within a cluster, instance failover within an Oracle Real Application Clusters (RAC) cluster, and failover and switchover between replicas.

See Also:

- *Oracle Data Guard Concepts and Administration* for more information about Oracle Data Guard
- *Oracle Database Oracle Clusterware and Oracle Real Application Clusters Administration and Deployment Guide*

Protection from Failures

RAC is the preferred method for protecting from an instance or system failure. After a failure, services are provided by a surviving node in the cluster. However, clustering does not protect from user error, media failure, or disasters. These types of failures require redundant copies of the database. You can make both physical and logical copies of a database.

Physical copies are identical, block for block, with the source database, and are the preferred means of protecting data. There are three types of physical copies: database backup, mirrored or multiplexed database files, and a physical standby database.

Logical copies contain the same information as the source database, but the information can be stored differently within the database. Creating a logical copy of your database offers many advantages. However, you should always create a logical copy in addition to a physical copy, not instead of physical copy.

A logical copy has the following benefits:

- A logical copy can be open while being updated. This ability makes the logical copy useful for near real-time reporting.
- A logical copy can have a different physical layout that is optimized for its own purpose. For example, it can contain additional indexes, and thereby improve the performance of reporting applications that utilize the logical copy.
- A logical copy provides better protection from corruptions. Because data is logically captured and applied, it is very unlikely that a physical corruption can propagate to the logical copy of the database.

There are three types of logical copies of a database:

- Logical standby databases
- Streams replica databases
- Application-maintained copies

Logical standby databases are best maintained using Oracle Data Guard in SQL apply mode. The rest of this chapter discusses Streams replica databases and application maintained copies.

See Also:

- *Oracle Database Backup and Recovery Basics* and *Oracle Database Backup and Recovery Advanced User's Guide* for more information about database backups and mirroring or multiplexing database files
- *Oracle Data Guard Concepts and Administration* for more information about physical standby databases and logical standby databases

Streams Replica Database

Like Oracle Data Guard in SQL apply mode, Oracle Streams can capture database changes, propagate them to destinations, and apply the changes at these destinations. Streams is optimized for replicating data. Streams can capture changes locally in the online redo log as it is written, and the captured changes can be propagated asynchronously to replica databases. This optimization can reduce the latency and can enable the replicas to lag the primary database by no more than a few seconds.

Nevertheless, you might choose to use Streams to configure and maintain a logical copy of your production database. Although using Streams might require additional work, it offers increased flexibility that might be required to meet specific business requirements. A logical copy configured and maintained using Streams is called a replica, not a logical standby, because it provides many capabilities that are beyond the scope of the normal definition of a standby database. Some of the requirements that can best be met using an Oracle Streams replica are listed in the following sections.

> **See Also:** *Oracle Streams Replication Administrator's Guide f*or more information about replicating database changes with Streams

Updates at the Replica Database

The greatest difference between a replica database and a standby database is that a replica database can be updated and a standby database cannot. Applications that must update data can run against the replica, including job queues and reporting applications that log reporting activity. Replica databases also allow local applications to operate autonomously, protecting local applications from WAN failures and reducing latency for database operations.

Heterogeneous Platform Support

The production and the replica do not need to be running on the exact same platform. This provides more flexibility in using computing assets, and facilitates migration between platforms.

Multiple Character Sets

Streams replicas can use different character sets than the production database. Data is automatically converted from one character set to another before being applied. This ability is extremely important if you have global operations and you must distribute data in multiple countries.

Mining the Online Redo Logs to Minimize Latency

If the replica is used for near real-time reporting, Streams can lag the production database by no more than a few seconds, providing up-to-date and accurate queries. Changes can be read from the online redo logs as the logs are written, rather than from the redo logs after archiving.

Greater than Ten Copies of Data

Streams supports unlimited numbers of replicas. Its flexible routing architecture allows for hub-and-spoke configurations that can efficiently propagate data to hundreds of replicas. This ability can be important if you must provide autonomous operation to many local offices in your organization. In contrast, because standby databases configured with Oracle Data Guard use the `LOG_ARCHIVE_DEST_`*n* initialization parameter to specify destinations, there is a limit of ten copies when you use Oracle Data Guard.

Fast Failover

Streams replicas can be open to read/write operations at all times. If a primary database fails, then Streams replicas are able to instantly resume processing. A small window of data might be left at the primary database, but this data will be automatically applied when the primary database recovers. This ability can be important if you value fast recovery time over no lost data. Assuming the primary database can eventually be recovered, the data is only temporarily unavailable.

Single Capture for Multiple Destinations

In a complex environment, changes need only be captured once. These changes can then be sent to multiple destinations. This ability enables more efficient use of the resources needed to mine the redo logs for changes.

When Not to Use Streams

As mentioned previously, there are scenarios in which you might choose to use Streams to meet some of your high availability requirements. One of the rules of high availability is to keep it simple. Oracle Data Guard is designed for high availability and is easier to implement than a Streams-based high availability solution. If you decide to leverage the flexibility offered by Streams, then you must be prepared to invest in the expertise and planning required to make a Streams-based solution robust. This means writing scripts to implement much of the automation and management tools provided with Oracle Data Guard.

Application-maintained Copies

The best availability can be achieved by designing the maintenance of logical copies of data directly into an application. The application knows what data is valuable and must be immediately moved off-site to guarantee no data loss. It can also synchronously replicate truly critical data, while asynchronously replicating less critical data. Applications maintain copies of data by either synchronously or asynchronously sending data to other applications that manage another logical copy of the data. Synchronous operations are performed using the distributed SQL or remote procedure features of the database. Asynchronous operations are performed using Advanced Queuing. Advanced Queuing is a database message queuing feature that is part of Oracle Streams.

Although the highest levels of availability can be achieved with application-maintained copies of data, great care is required to realize these results. Typically, a great amount of custom development is required. Many of the difficult boundary conditions that have been analyzed and solved with solutions such as Oracle Data Guard and Streams **replication** must be reanalyzed and solved by the custom application developers. In addition, standard solutions like Oracle Data Guard and Streams replication undergo stringent testing both by Oracle and its customers. It will take a great deal of effort before a custom-developed solution can exhibit the same degree of maturity. For these reasons, only organizations with substantial patience and expertise should attempt to build a high availability solution with application maintained copies.

> **See Also:** *Oracle Streams Advanced Queuing User's Guide and Reference* for more information about developing applications with Advanced Queuing

Best Practices for Streams High Availability Environments

Implementing Streams in a high availability environment requires consideration of possible failure and recovery scenarios, and the implementation of procedures to ensure Streams continues to capture, propagate, and apply changes after a failure. Some of the issues that must be examined include the following:

- Configuring Streams for High Availability
  - Directly Connecting Every Database to Every Other Database
  - Creating Hub-and-Spoke Configurations
  - Configuring Oracle Real Application Clusters with Streams
  - Local or Downstream Capture with Streams
- Recovering from Failures
  - Automatic Capture Process Restart After a Failover
  - Database Links Reestablishment After a Failover
  - Propagation Job Restart After a Failover
  - Automatic Apply Process Restart After a Failover

The following sections discuss these issues in detail.

Configuring Streams for High Availability

When configuring a solution using Streams, it is important to anticipate failures and design availability into the architecture. You must examine every database in the distributed system, and design a recovery plan in case of failure of that database. In some situations, failure of a database affects only services accessing data on that database. In other situations, a failure is multiplied, because it can affect other databases.

Directly Connecting Every Database to Every Other Database

A configuration where each database is directly connected to every other database in the distributed system is the most resilient to failures, because a failure of one database will not prevent any other databases from operating or communicating. Assuming all data is replicated, services that were using the failed database can connect to surviving replicas.

See Also:

- *Oracle Streams Replication Administrator's Guide* for a detailed example of such an environment
- "Queue Forwarding and Apply Forwarding" on page 3-7

Creating Hub-and-Spoke Configurations

Although configurations where each database is directly connected to every other database provide the best high availability characteristics, they can become difficult to manage when the number of databases becomes large. Hub-and-spoke configurations solve this manageability issue by funneling changes from many databases into a hub database, and then to other hub databases, or to other spoke databases. To add a new source or destination, you simply connect it to a hub database, rather than establishing connections to every other database.

A hub, however, becomes a very important node in your distributed environment. Should it fail, all communications flowing through the hub will fail. Due to the asynchronous nature of the **message**s propagating through the hub, it can be very difficult to redirect a stream from one hub to another. A better approach is to make the hub resilient to failures.

The same techniques used to make a single database resilient to failures also apply to distributed hub databases. Oracle recommends RAC to provide protection from instance and node failures. This configuration should be combined with a "no loss" physical standby database, to protect from disasters and data errors. Oracle does not recommend using a Streams replica as the only means to protect from disasters or data errors.

> **See Also:** *Oracle Streams Replication Administrator's Guide* for a detailed example of such an environment

Configuring Oracle Real Application Clusters with Streams

Using RAC with Streams introduces some important considerations. When running in a RAC cluster, a **capture process** runs on the instance that owns the **queue** that is receiving the captured logical change records (LCRs). Job queues should be running on all instances, and a **propagation job** running on an instance will propagate LCRs from any queue owned by that instance to **destination queue**s. An **apply process** runs on the instance that owns the queue from which the apply process dequeues its messages. That might or might not be the same queue on which capture runs.

Any propagation to the database running RAC is made over database links. The database links must be configured to connect to the destination instance that owns the queue that will receive the messages.

You might choose to use a cold failover cluster to protect from system failure rather than RAC. A cold failover cluster is not RAC. Instead, a cold failover cluster uses a secondary node to mount and recover the database when the first node fails.

> **See Also:**
>
> - "Streams Capture Processes and Oracle Real Application Clusters" on page 2-21
> - "Queues and Oracle Real Application Clusters" on page 3-13
> - "Streams Apply Processes and Oracle Real Application Clusters" on page 4-9

Local or Downstream Capture with Streams

Beginning in Oracle Database 10*g*, Streams supports capturing changes from the redo log on the local **source database** or at a **downstream database** at a different site. The choice of local capture or downstream capture has implications for availability. When a failure occurs at a source database, some changes might not have been captured. With local capture, those changes might not be available until the source database is recovered. In the event of a catastrophic failure, those changes might be lost.

Downstream capture at a remote database reduces the window of potential data loss in the event of a failure. Depending on the configuration, downstream capture enables you to guarantee all changes committed at the source database are safely copied to a remote site, where they can be captured and propagated to other databases and applications. Streams uses the same mechanism as Oracle Data Guard to copy redo data or log files to remote destinations, and supports the same operational modes, including maximum protection, maximum availability, and maximum performance.

See Also: "Local Capture and Downstream Capture" on page 2-12

Recovering from Failures

The following sections provide best practices for recovering from failures.

Automatic Capture Process Restart After a Failover

After a failure and restart of a single-node database, or a failure and restart of a database on another node in a cold failover cluster, the **capture process** automatically returns to the status it was in at the time of the failure. That is, if it was running at the time of the failure, then the capture process restarts automatically.

Similarly, for a capture process running in a RAC environment, if an instance running the capture process fails, then the **queue** that receives the **captured message**s is assigned to another node in the cluster, and the capture process is restarted automatically. A capture process follows its queue to a different instance if the current owner instance becomes unavailable, and the queue itself follows the rules for primary instance and secondary instance ownership.

See Also:

- "Streams Capture Processes and Oracle Real Application Clusters" on page 2-21
- "Starting a Capture Process" on page 11-24
- "Queues and Oracle Real Application Clusters" on page 3-13 for information about primary and secondary instance ownership for queues

Database Links Reestablishment After a Failover

It is important to ensure that a **propagation** continues to function after a failure of a **destination database** instance. A **propagation job** will retry (with increasing delay between retries) its database link sixteen times after a failure until the connection is reestablished. If the connection is not reestablished after sixteen tries, then the **propagation schedule** is disabled.

If the database is restarted on the same node, or on a different node in a cold failover cluster, then the connection should be reestablished. In some circumstances, the database link could be waiting on a read or write, and will not detect the failure until a lengthy timeout expires. The timeout is controlled by the `TCP_KEEPALIVE_INTERVAL` TCP/IP parameter. In such circumstances, you should drop and re-create the database link to ensure that communication is reestablished quickly.

When an instance in a RAC cluster fails, the instance is recovered by another node in the cluster. Each queue that was previously owned by the failed instance is assigned to a new instance. If the failed instance contained one or more **destination queue**s for propagations, then queue-to-queue propagations automatically failover to the new instance. However, for queue-to-dblink propagations, you must drop and reestablish any inbound database links to point to the new instance that owns a destination queue. You do not need to modify a propagation that uses a re-created database link.

In a high availability environment, you can prepare scripts that will drop and re-create all necessary database links. After a failover, you can execute these scripts so that Streams can resume propagation.

See Also:

- "Configuring a Streams Administrator" on page 10-1 for information about creating database links in a Streams environment
- "Queues and Oracle Real Application Clusters" on page 3-13 for more information about database links in a RAC environment

Propagation Job Restart After a Failover

For **message**s to be propagated from a **source queue** to a **destination queue**, a **propagation job** must run on the instance owning the source queue. In a single-node database, or cold failover cluster, propagation resumes when the single database instance is restarted.

When running in a RAC environment, a propagation job runs on the instance that owns the source queue from which the propagation job sends messages to a destination queue. If the owner instance for a propagation job goes down, then the propagation job automatically migrates to a new owner instance. You should not alter instance affinity for Streams propagation jobs, because Streams manages instance affinity for propagation jobs automatically. Also, for any jobs to run on an instance, the modifiable initialization parameter `JOB_QUEUE_PROCESSES` must be greater than zero for that instance.

See Also: "Queues and Oracle Real Application Clusters" on page 3-13

Automatic Apply Process Restart After a Failover

After a failure and restart of a single-node database, or a failure and restart of a database on another node in a cold failover cluster, the apply process automatically returns to the status it was in at the time of the failure. That is, if it was running at the time of the failure, then the apply process restarts automatically.

Similarly, in a RAC cluster, if an instance hosting the apply process fails, then the queue from which the apply process dequeues messages is assigned to another node in the cluster, and the apply process is restarted automatically. An apply process follows its queue to a different instance if the current owner instance becomes unavailable, and the queue itself follows the rules for primary instance and secondary instance ownership.

See Also:

- "Streams Apply Processes and Oracle Real Application Clusters" on page 4-9
- "Starting an Apply Process" on page 13-7
- "Queues and Oracle Real Application Clusters" on page 3-13 for information about primary and secondary instance ownership for queues

Part II

Streams Administration

This part describes managing a Streams environment, including step-by-step instructions for configuring, administering, monitoring and troubleshooting. This part contains the following chapters:

10

Preparing a Streams Environment

This chapter provides instructions for preparing a database or a distributed database environment to use Streams.

This chapter contains these topics:

Configuring a Streams Administrator

To manage a Streams environment, either create a new user with the appropriate privileges or grant these privileges to an existing user. You should not use the `SYS` or `SYSTEM` user as a Streams administrator, and the Streams administrator should not use the `SYSTEM` tablespace as its default tablespace.

Complete the following steps to configure a Streams administrator at each database in the environment that will use Streams:

1. Connect in SQL*Plus as an administrative user who can create users, grant privileges, and create tablespaces. Remain connected as this administrative user for all subsequent steps.
2. Either create a tablespace for the Streams administrator or use an existing tablespace. For example, the following statement creates a new tablespace for the Streams administrator:

```
CREATE TABLESPACE streams_tbs DATAFILE '/usr/oracle/dbs/streams_tbs.dbf'
  SIZE 25M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;
```

3. Create a new user to act as the Streams administrator or use an existing user. For example, to create a new user named `strmadmin` and specify that this user uses the `streams_tbs` tablespace, run the following statement:

```
CREATE USER strmadmin
 IDENTIFIED BY strmadminpw
   DEFAULT TABLESPACE streams_tbs
   QUOTA UNLIMITED ON streams_tbs;
```

> **Note:** For security purposes, use a password other than `strmadminpw` for the Streams administrator.

4. Grant the Streams administrator DBA role:

```
GRANT DBA TO strmadmin;
```

5. Optionally, run the GRANT_ADMIN_PRIVILEGE procedure in the DBMS_STREAMS_AUTH package. You might choose to run this procedure on the Streams administrator created in Step3 if any of the following conditions are true:

   - The Streams administrator will run user-created subprograms that execute subprograms in Oracle-supplied packages associated with Streams. An example is a user-created stored procedure that executes a procedure in the DBMS_STREAMS_ADM package.
   - The Streams administrator will run user-created subprograms that query data dictionary views associated with Streams. An example is a user-created stored procedure that queries the DBA_APPLY_ERROR data dictionary view.

   A user must have explicit EXECUTE privilege on a package to execute a subprogram in the package inside of a user-created subprogram, and a user must have explicit SELECT privilege on a data dictionary view to query the view inside of a user-created subprogram. These privileges cannot be through a role. You can run the GRANT_ADMIN_PRIVILEGE procedure to grant such privileges to the Streams administrator, or you can grant them directly.

   Depending on the parameter settings for the GRANT_ADMIN_PRIVILEGE procedure, it either grants the privileges needed to be a Streams administrator directly, or it generates a script that you can edit and then run to grant these privileges.

   See Also: *Oracle Database PL/SQL Packages and Types Reference* for more information about this procedure

   Use the GRANT_ADMIN_PRIVILEGE procedure to grant privileges directly:

```
BEGIN
  DBMS_STREAMS_AUTH.GRANT_ADMIN_PRIVILEGE(
    grantee          => 'strmadmin',
    grant_privileges => true);
END;
/
```

   Use the GRANT_ADMIN_PRIVILEGE procedure to generate a script:

   a. Use the SQL statement CREATE DIRECTORY to create a directory object for the directory into which you want to generate the script. A directory object is similar to an alias for the directory. For example, to create a directory object called admin_dir for the /usr/admin directory on your computer system, run the following procedure:

```
CREATE DIRECTORY admin_dir AS '/usr/admin';
```

   b. Run the GRANT_ADMIN_PRIVILEGE procedure to generate a script named grant_strms_privs.sql and place this script in the /usr/admin directory on your computer system:

```
BEGIN
  DBMS_STREAMS_AUTH.GRANT_ADMIN_PRIVILEGE(
    grantee          => 'strmadmin',
    grant_privileges => false,
    file_name        => 'grant_strms_privs.sql',
    directory_name   => 'admin_dir');
END;
/
```

Notice that the `grant_privileges` parameter is set to `false` so that the procedure does not grant the privileges directly. Also, notice that the directory object created in Step a is specified for the `directory_name` parameter.

c. Edit the generated script if necessary and save your changes.

d. Execute the script in SQL*Plus:

```
SET ECHO ON
SPOOL grant_strms_privs.out
@/usr/admin/grant_strms_privs.sql
SPOOL OFF
```

e. Check the spool file to ensure that all of the grants executed successfully. If there are errors, then edit the script to correct the errors and rerun it.

6. If necessary, grant the Streams administrator the following privileges:

- If no **apply user** is specified for an apply process, then the necessary privileges to perform DML and DDL changes on the apply objects owned by another user. If an apply user is specified, then the apply user must have these privileges.
- If no apply user is specified for an apply process, then `EXECUTE` privilege on any PL/SQL procedure owned by another user that is executed by a Streams apply process. These procedures can be used in **apply handler**s or **error handler**s. If an apply user is specified, then the apply user must have these privileges.
- `EXECUTE` privilege on any PL/SQL function owned by another user that is specified in a **custom rule-based transformation** for a **rule** used by a Streams **capture process**, **propagation**, **apply process**, or **messaging client**. For a capture process, if a **capture user** is specified, then the capture user must have these privileges. For an apply process, if an apply user is specified, then the apply user must have these privileges.
- Privileges to alter database objects where appropriate. For example, if the Streams administrator must create a **supplemental log group** for a table in another schema, then the Streams administrator must have the necessary privileges to alter the table.
- If the Streams administrator does not own the **queue** used by a Streams capture process, propagation, apply process, or messaging client, and is not specified as the queue user for the queue when the queue is created, then the Streams administrator must be configured as a **secure queue** user of the queue if you want the Streams administrator to be able to enqueue **message**s into or dequeue messages from the queue. The Streams administrator might also need `ENQUEUE` or `DEQUEUE` privileges on the queue, or both. See "Enabling a User to Perform Operations on a Secure Queue" on page 12-3 for instructions.
- `EXECUTE` privilege on any object types that the Streams administrator might need to access.

7. Repeat all of the previous steps at each database in the environment that will use Streams.

See Also: "Monitoring Streams Administrators and Other Streams Users" on page 26-1

Setting Initialization Parameters Relevant to Streams

Table 10–1 lists initialization parameters that are important for the operation, reliability, and performance of a Streams environment. Set these parameters appropriately for your Streams environment. This table specifies whether each parameter is modifiable. A modifiable initialization parameter can be modified using the `ALTER SYSTEM` statement while an instance is running. Some of the modifiable parameters can also be modified for a single session using the `ALTER SESSION` statement.

Table 10–1 Initialization Parameters Relevant to Streams

Parameter	Values	Description
`COMPATIBLE`	**Default:** `10.0.0` **Range:** `9.2.0` to Current Release Number **Modifiable?:** No	This parameter specifies the release with which the Oracle server must maintain compatibility. Oracle servers with different compatibility levels can interoperate. To use the new Streams features introduced in Oracle Database 10*g* Release 1, this parameter must be set to `10.1.0` or higher. To use downstream capture, this parameter must be set to `10.1.0` or higher at both the **source database** and the **downstream database**. To use the new Streams features introduced in Oracle Database 10*g* Release 2, this parameter must be set to `10.2.0` or higher.
`GLOBAL_NAMES`	**Default:** `false` **Range:** `true` or `false` **Modifiable?:** Yes	Specifies whether a database link is required to have the same name as the database to which it connects. To use Streams to share information between databases, set this parameter to `true` at each database that is participating in your Streams environment.
`JOB_QUEUE_PROCESSES`	**Default:** `0` **Range:** `0` to `1000` **Modifiable?:** Yes	Specifies the number of `Jn` job queue processes for each instance (`J000` ... `J999`). Job queue processes handle requests created by `DBMS_JOB`. This parameter must be set to at least `2` at each database that is propagating **message**s in your Streams environment, and should be set to the same value as the maximum number of jobs that can run simultaneously plus two.

Table 10–1 (Cont.) Initialization Parameters Relevant to Streams

Parameter	Values	Description
LOG_ARCHIVE_CONFIG	**Default:** 'SEND, RECEIVE, NODG_CONFIG' **Range**: Values: ▪ SEND ▪ NOSEND ▪ RECEIVE ▪ NORECEIVE ▪ DG_CONFIG ▪ NODG_CONFIG **Modifiable?:** Yes	Enables or disables the sending of redo logs to remote destinations and the receipt of remote redo logs, and specifies the unique database names (DB_UNIQUE_NAME) for each database in the Data Guard configuration To use downstream capture and copy the redo data to the downstream database using redo transport services, you can use the default setting for this parameter. If this parameter is set to a value other than the default, then make sure it includes the SEND value at the source database and the RECEIVE value at the downstream database.
LOG_ARCHIVE_DEST_*n*	**Default:** None **Range**: None **Modifiable?:** Yes	Defines up to ten log archive destinations, where *n* is 1, 2, 3, ... 10. To use downstream capture and copy the redo data to the downstream database using redo transport services, at least one log archive destination must be at the site running the **downstream capture process**.
LOG_ARCHIVE_DEST_STATE_*n*	**Default:** enable **Range**: One of the following: ▪ alternate ▪ reset ▪ defer ▪ enable **Modifiable?:** Yes	Specifies the availability state of the corresponding destination. The parameter suffix (1 through 10) specifies one of the ten corresponding LOG_ARCHIVE_DEST_*n* destination parameters. To use downstream capture and copy the redo data to the downstream database using redo transport services, make sure the destination that corresponds to the LOG_ARCHIVE_DEST_*n* destination for the downstream database is set to enable.
OPEN_LINKS	**Default:** 4 **Range:** 0 to 255 **Modifiable?:** No	Specifies the maximum number of concurrent open connections to remote databases in one session. These connections include database links, as well as external procedures and cartridges, each of which uses a separate process. In a Streams environment, make sure this parameter is set to the default value of 4 or higher.
PARALLEL_MAX_SERVERS	**Default:** Derived automatically **Range:** 0 to 3599 **Modifiable?:** Yes	Specifies the maximum number of parallel execution processes and parallel recovery processes for an instance. As demand increases, Oracle will increase the number of processes from the number created at instance startup up to this value. In a Streams environment, each **capture process** and **apply process** can use multiple parallel execution servers. Set this initialization parameter to an appropriate value to ensure that there are enough parallel execution servers.

Table 10–1 (Cont.) Initialization Parameters Relevant to Streams

Parameter	Values	Description
PROCESSES	**Default:** 40 to operating system-dependent **Range:** 6 to operating system-dependent **Modifiable?:** No	Specifies the maximum number of operating system user processes that can simultaneously connect to Oracle. Make sure the value of this parameter allows for all background processes, such as locks, job queue processes, and parallel execution processes. In Streams, capture processes and apply processes use background processes and parallel execution processes, and **propagation jobs** use job queue processes.
SESSIONS	**Default:** Derived from: (1.1 * PROCESSES) + 5 **Range:** 1 to 231 **Modifiable?:** No	Specifies the maximum number of sessions that can be created in the system. To run one or more capture processes or apply processes in a database, you might need to increase the size of this parameter. Each background process in a database requires a session.
SGA_MAX_SIZE	**Default:** Initial size of SGA at startup **Range:** 0 to operating system-dependent **Modifiable?:** No	Specifies the maximum size of SGA for the lifetime of a database instance. To run multiple capture processes on a single database, you might need to increase the size of this parameter.
SGA_TARGET	**Default:** 0 (SGA autotuning is disabled) **Range:** 64 to operating system-dependent **Modifiable?:** Yes	Specifies the total size of all System Global Area (SGA) components. If this parameter is set to a nonzero value, then the size of the **Streams pool** is managed by Automatic Shared Memory Management.
SHARED_POOL_SIZE	**Default:** If SGA_TARGET is set: If the parameter is not specified, then the default is 0 (internally determined by the Oracle Database). If the parameter is specified, then the user-specified value indicates a minimum value for the memory pool. If SGA_TARGET is not set (32-bit platforms): 32 MB, rounded up to the nearest granule size If SGA_TARGET is not set (64-bit platforms): 84 MB, rounded up to the nearest granule size **Range:** The granule size to operating system-dependent **Modifiable?:** Yes	Specifies (in bytes) the size of the shared pool. The shared pool contains shared cursors, stored procedures, control structures, and other structures. If the SGA_TARGET and STREAMS_POOL_SIZE initialization parameters are set to zero, then Streams transfers an amount equal to 10% of the shared pool from the buffer cache to the Streams pool.

Table 10–1 (Cont.) Initialization Parameters Relevant to Streams

Parameter	Values	Description
STREAMS_POOL_SIZE	**Default:** 0 **Range:** Minimum: 0 Maximum: operating system-dependent **Modifiable?:** Yes	Specifies (in bytes) the size of the Streams pool. The Streams pool contains **buffered queue** messages. In addition, the Streams pool is used for internal communications during parallel capture and apply. If the SGA_TARGET initialization parameter is set to a nonzero value, then the Streams pool size is set by Automatic Shared memory management, and STREAMS_POOL_SIZE specifies the minimum size. This parameter is modifiable. If this parameter is reduced to zero when an instance is running, then Streams processes and jobs will not run. You should increase the size of the Streams pool for each of the following factors: ▪ 10 MB for each capture process parallelism ▪ 10 MB or more for each buffered queue ▪ 1 MB for each apply process parallelism You can use the V$STREAMS_POOL_ADVICE dynamic performance view to determine an appropriate setting for this parameter. **See Also:** "Streams Pool" on page 3-19
TIMED_STATISTICS	**Default:** If STATISTICS_LEVEL is set to TYPICAL or ALL, then true If STATISTICS_LEVEL is set to BASIC, then false The default for STATISTICS_LEVEL is TYPICAL. **Range:** true or false **Modifiable?:** Yes	Specifies whether or not statistics related to time are collected. To collect elapsed time statistics in the dynamic performance views related to Streams, set this parameter to true. The views that include elapsed time statistics include: V$STREAMS_CAPTURE, V$STREAMS_APPLY_COORDINATOR, V$STREAMS_APPLY_READER, V$STREAMS_APPLY_SERVER.
UNDO_RETENTION	**Default:** 900 **Range:** 0 to 2^{32}-1 (max value represented by 32 bits) **Modifiable?:** Yes	Specifies (in seconds) the amount of committed undo information to retain in the database. For a database running one or more capture processes, make sure this parameter is set to specify an adequate undo retention period. If you are running one or more capture processes and you are unsure about the proper setting, then try setting this parameter to at least 3600. If you encounter "snapshot too old" errors, then increase the setting for this parameter until these errors cease. Make sure the undo tablespace has enough space to accommodate the UNDO_RETENTION setting.

See Also:

- *Oracle Database Reference* for more information about these initialization parameters
- *Oracle Data Guard Concepts and Administration* for more information about the `LOG_ARCHIVE_DEST_n` parameter
- *Oracle Database Administrator's Guide* for more information about the `UNDO_RETENTION` parameter

Configuring Network Connectivity and Database Links

If you plan to use Streams to share information between databases, then configure network connectivity and database links between these databases:

- For Oracle databases, configure your network and Oracle Net so that the databases can communicate with each other.

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

- For non-Oracle databases, configure an Oracle gateway for communication between the Oracle database and the non-Oracle database.

  See Also: *Oracle Database Heterogeneous Connectivity Administrator's Guide*

- If you plan to propagate **message**s from a **source queue** at a database to a **destination queue** at another database, then create a private database link between the database containing the source queue and the database containing the destination queue. Each database link should use a `CONNECT TO` clause for the user propagating messages between databases.

  For example, to create a database link to a database named `dbs2.net` connecting as a Streams administrator named `strmadmin`, run the following statement:

  ```
  CREATE DATABASE LINK dbs2.net CONNECT TO strmadmin IDENTIFIED BY strmadminpw
     USING 'dbs2.net';
  ```

  See Also: *Oracle Database Administrator's Guide* for more information about creating database links

11

Managing a Capture Process

A **capture process** captures changes in a redo log, reformats the captured changes into logical change records (LCRs), and enqueues the LCRs into an `ANYDATA` queue.

This chapter contains these topics:

- Creating a Capture Process
- Starting a Capture Process
- Stopping a Capture Process
- Managing the Rule Set for a Capture Process
- Setting a Capture Process Parameter
- Setting the Capture User for a Capture Process
- Managing the Checkpoint Retention Time for a Capture Process
- Specifying Supplemental Logging at a Source Database
- Adding an Archived Redo Log File to a Capture Process Explicitly
- Setting the First SCN for an Existing Capture Process
- Setting the Start SCN for an Existing Capture Process
- Specifying Whether Downstream Capture Uses a Database Link
- Managing Extra Attributes in Captured Messages
- Dropping a Capture Process

Each task described in this chapter should be completed by a Streams administrator that has been granted the appropriate privileges, unless specified otherwise.

See Also:

- Chapter 2, "Streams Capture Process"
- "Configuring a Streams Administrator" on page 10-1

Creating a Capture Process

You can create a **capture process** that captures changes either locally at the **source database** or remotely at a **downstream database**. If a capture process runs on a downstream database, then redo data from the source database is copied to the downstream database, and the capture process captures changes in redo data at the downstream database.

You can use any of the following procedures to create a **local capture process**:

- `DBMS_STREAMS_ADM.ADD_TABLE_RULES`
- `DBMS_STREAMS_ADM.ADD_SUBSET_RULES`
- `DBMS_STREAMS_ADM.ADD_SCHEMA_RULES`
- `DBMS_STREAMS_ADM.ADD_GLOBAL_RULES`
- `DBMS_CAPTURE_ADM.CREATE_CAPTURE`

Each of the procedures in the `DBMS_STREAMS_ADM` package creates a capture process with the specified name if it does not already exist, creates either a positive or **negative rule set** for the capture process if the capture process does not have such a **rule set**, and can add **table rule**s, **schema rule**s, or **global rule**s to the rule set.

The `CREATE_CAPTURE` procedure creates a capture process, but does not create a rule set or rules for the capture process. However, the `CREATE_CAPTURE` procedure enables you to specify an existing rule set to associate with the capture process, either as a positive or a negative rule set, a **first SCN**, and a **start SCN** for the capture process. To create a capture process that performs downstream capture, you must use the `CREATE_CAPTURE` procedure.

Attention: When a capture process is started or restarted, it might need to scan redo log files with a `FIRST_CHANGE#` value that is lower than start SCN. Removing required redo log files before they are scanned by a capture process causes the capture process to abort. You can query the `DBA_CAPTURE` data dictionary view to determine the first SCN, start SCN, and **required checkpoint SCN** for a capture process. A capture process needs the redo log file that includes the required checkpoint SCN, and all subsequent redo log files. See "Capture Process Creation" on page 2-27 for more information about the first SCN and start SCN for a capture process.

Note: To configure downstream capture, the source database must be an Oracle Database 10*g* Release 1 database or later.

The following sections describe:

- Preparing to Create a Capture Process
- Creating a Local Capture Process
- Preparing for and Creating a Real-Time Downstream Capture Process
- Creating an Archived-Log Downstream Capture Process that Assigns Logs Implicitly
- Creating an Archived-Log Downstream Capture Process that Assigns Logs Explicitly
- Creating a Local Capture Process with Non-NULL Start SCN

Note:

- After creating a capture process, avoid changing the `DBID` or global name of the source database for the capture process. If you change either the `DBID` or global name of the source database, then the capture process must be dropped and re-created.
- To create a capture process, a user must be granted `DBA` role.

See Also:

- "Capture Process Creation" on page 2-27
- "First SCN and Start SCN" on page 2-19
- *Oracle Streams Replication Administrator's Guide* for information about changing the DBID or global name of a source database

Preparing to Create a Capture Process

The following tasks must be completed before you create a capture process:

- Configure any **source database** that generates redo data that will be captured by a capture process to run in `ARCHIVELOG` mode. See "ARCHIVELOG Mode and a Capture Process" on page 2-38 and *Oracle Database Administrator's Guide*. For **downstream capture process**es, the **downstream database** also must run in `ARCHIVELOG` mode if you plan to configure a **real-time downstream capture process**. The downstream database does not need to run in `ARCHIVELOG` mode if you plan to run only **archived-log downstream capture process** on it.
- Make sure the initialization parameters are set properly on any database that will run a capture process. See "Setting Initialization Parameters Relevant to Streams" on page 10-4.
- Create a Streams administrator on each database involved in the Streams configuration. See "Configuring a Streams Administrator" on page 10-1. The examples in this chapter assume that the Streams administrator is `strmadmin`.
- Create an `ANYDATA` queue to associate with the capture process, if one does not exist. See "Creating an ANYDATA Queue" on page 12-2 for instructions. The examples in this chapter assume that the **queue** used by the capture process is `strmadmin.streams_queue`. Create the queue on the same database that will run the capture process.

Creating a Local Capture Process

The following sections describe using the DBMS_STREAMS_ADM package and the DBMS_CAPTURE_ADM package to create a **local capture process**. Make sure you complete the tasks in "Preparing to Create a Capture Process" on page 11-3 before you proceed.

Example of Creating a Local Capture Process Using DBMS_STREAMS_ADM

The following example runs the ADD_TABLE_RULES procedure in the DBMS_STREAMS_ADM package to create a local capture process:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name         => 'hr.employees',
    streams_type       => 'capture',
    streams_name       => 'strm01_capture',
    queue_name         => 'strmadmin.streams_queue',
    include_dml        => true,
    include_ddl        => true,
    include_tagged_lcr => false,
    source_database    => NULL,
    inclusion_rule     => true);
END;
/
```

Running this procedure performs the following actions:

- Creates a capture process named strm01_capture. The capture process is created only if it does not already exist. If a new capture process is created, then this procedure also sets the **start SCN** to the point in time of creation.
- Associates the capture process with an existing **queue** named streams_queue.
- Creates a **positive rule set** and associates it with the capture process, if the capture process does not have a positive rule set, because the inclusion_rule parameter is set to true. The rule set uses the SYS.STREAMS$_EVALUATION_CONTEXT **evaluation context**. The rule set name is system generated.
- Creates two **rule**s. One rule evaluates to TRUE for DML changes to the hr.employees table, and the other rule evaluates to TRUE for DDL changes to the hr.employees table. The rule names are system generated.
- Adds the two rules to the positive rule set associated with the capture process. The rules are added to the positive rule set because the inclusion_rule parameter is set to true.
- Specifies that the capture process captures a change in the redo log only if the change has a NULL **tag**, because the include_tagged_lcr parameter is set to false. This behavior is accomplished through the **system-created rule**s for the capture process.
- Creates a capture process that captures local changes to the **source database** because the source_database parameter is set to NULL. For a local capture process, you can also specify the global name of the local database for this parameter.
- Prepares the hr.employees table for **instantiation**.

See Also:

- "Capture Process Creation" on page 2-27
- "System-Created Rules" on page 6-5
- *Oracle Streams Replication Administrator's Guide* for more information about Streams tags

Example of Creating a Local Capture Process Using DBMS_CAPTURE_ADM

The following example runs the `CREATE_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package to create a local capture process:

```
BEGIN
  DBMS_CAPTURE_ADM.CREATE_CAPTURE(
    queue_name          => 'strmadmin.streams_queue',
    capture_name        => 'strm02_capture',
    rule_set_name       => 'strmadmin.strm01_rule_set',
    start_scn           => NULL,
    source_database     => NULL,
    use_database_link   => false,
    first_scn           => NULL);
END;
/
```

Running this procedure performs the following actions:

- Creates a capture process named `strm02_capture`. A capture process with the same name must not exist.
- Associates the capture process with an existing queue named `streams_queue`.
- Associates the capture process with an existing rule set named `strm01_rule_set`. This rule set is the positive rule set for the capture process.
- Creates a capture process that captures local changes to the source database because the `source_database` parameter is set to `NULL`. For a local capture process, you can also specify the global name of the local database for this parameter.
- Specifies that the Oracle database determines the start SCN and **first SCN** for the capture process because both the `start_scn` parameter and the `first_scn` parameter are set to `NULL`.
- If no other capture processes that capture local changes are running on the local database, then the `BUILD` procedure in the `DBMS_CAPTURE_ADM` package is run automatically. Running this procedure extracts the data dictionary to the redo log, and a **LogMiner data dictionary** is created when the capture process is started for the first time.

See Also:

- "Capture Process Creation" on page 2-27
- "SCN Values Relating to a Capture Process" on page 2-19

Preparing for and Creating a Real-Time Downstream Capture Process

To create a capture process that performs downstream capture, you must use the `CREATE_CAPTURE` procedure. The example in this section describes creating a **real-time downstream capture process** that uses a database link to the **source database**. However, a real-time downstream capture process might not use a database link.

This example assumes the following:

- The source database is `dbs1.net` and the **downstream database** is `dbs2.net`.
- The capture process that will be created at `dbs2.net` uses the `streams_queue`.
- The capture process will capture DML changes to the `hr.departments` table.

> **See Also:** "Downstream Capture" on page 2-13 for conceptual information about real-time downstream capture

Preparing to Copy Redo Data for Real-Time Downstream Capture

Complete the following steps to prepare the source database to copy its redo data to the downstream database, and to prepare the downstream database to accept the redo data:

1. Complete the tasks in "Preparing to Create a Capture Process" on page 11-3.
2. Configure Oracle Net so that the source database can communicate with the downstream database.

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

3. At the source database, set the following initialization parameters to configure redo transport services to use the log writer process (LGWR) to copy redo data from the online redo log at the source database to the standby redo log at the downstream database:
   - Set at least one archive log destination in the `LOG_ARCHIVE_DEST_`*n* initialization parameter to the computer system running the downstream database. To do this, set the following attributes of this parameter:
     - `SERVICE` - Specify the network service name of the downstream database.
     - `LGWR ASYNC` or `LGWR SYNC` - Specify this attribute so that the log writer process (LGWR) will send redo data to the downstream database.

       When you specify `LGWR ASYNC`, network I/O is performed asynchronously for the destination. Therefore, the LGWR process submits the network I/O request for the destination and continues processing the next request without waiting for the I/O to complete and without checking the completion status of the I/O. The advantage of specifying `LGWR ASYNC` is that it results in little or no effect on the performance of the source database. If the source database is running Oracle Database 10*g* Release 1 or later, then `LGWR ASYNC` is recommended to avoid affecting source database performance if the downstream database or network is performing poorly.

When you specify LGWR SYNC, network I/O is performed synchronously for the destination, which means that once the I/O is initiated, the LGWR process waits for the I/O to complete before continuing. The advantage of specifying LGWR SYNC attribute is that redo log files are sent to the downstream database faster then when LGWR ASYNC is specified. Also, specifying LGWR SYNC AFFIRM results in behavior that is similar to MAXIMUM AVAILABILITY standby protection mode. Note that specifying an ALTER DATABASE STANDBY DATABASE TO MAXIMIZE AVAILABILITY SQL statement has no effect on a Streams capture process.

- MANDATORY or OPTIONAL - If you specify MANDATORY, then archiving of a redo log file to the downstream database must succeed before the corresponding online redo log at the source database can be overwritten. If you specify OPTIONAL, then successful archiving of a redo log file to the downstream database is not required before the corresponding online redo log at the source database can be overwritten. Either MANDATORY or OPTIONAL is acceptable for a downstream database destination. If neither the MANDATORY nor the OPTIONAL attribute is specified, then the default is OPTIONAL.
- NOREGISTER - Specify this attribute so that the downstream database location is not recorded in the downstream database control file.
- VALID FOR - Specify either (ONLINE_LOGFILE,PRIMARY_ROLE) or (ONLINE_LOGFILE,ALL_ROLES).

The following example is a LOG_ARCHIVE_DEST_*n* setting at the source database that specifies a real-time capture downstream database:

```
LOG_ARCHIVE_DEST_2='SERVICE=DBS2.NET LGWR ASYNC OPTIONAL NOREGISTER
   VALID_FOR=(ONLINE_LOGFILE,PRIMARY_ROLE)'
```

You can specify other attributes in the LOG_ARCHIVE_DEST_*n* initialization parameter if necessary.

- Set the LOG_ARCHIVE_DEST_STATE_*n* initialization parameter that corresponds with the LOG_ARCHIVE_DEST_*n* parameter for the downstream database to ENABLE.

  For example, if the LOG_ARCHIVE_DEST_2 initialization parameter is set for the downstream database, then set one LOG_ARCHIVE_DEST_STATE_2 parameter in the following way:

  ```
  LOG_ARCHIVE_DEST_STATE_2=ENABLE
  ```

- Make sure the setting for the LOG_ARCHIVE_CONFIG initialization parameter includes the send value.

See Also: *Oracle Database Reference* and *Oracle Data Guard Concepts and Administration* for more information about these initialization parameters

4. At the downstream database, set the following initialization parameters to configure the downstream database to receive redo data from the source database LGWR and write the redo data to the standby redo log at the downstream database:

   - Set at least one archive log destination in the LOG_ARCHIVE_DEST_*n* initialization parameter to a directory on the computer system running the downstream database. To do this, set the following attributes of this parameter:
     - LOCATION - Specify a valid path name for a disk directory on the system that hosts the downstream database. Each destination that specifies the LOCATION attribute must identify a unique directory path name. This is the local destination for archived redo log files written from the standby redo logs. Log files from a remote source database should be kept separate from local database log files. In addition, if the downstream database contains log files from multiple source databases, then the log files from each source database should be kept separate from each other.
     - MANDATORY - Successful archiving of a standby redo log file must succeed before the corresponding standby redo log file can be overwritten.
     - VALID FOR - Specify either (STANDBY_LOGFILE,PRIMARY_ROLE) or (STANDBY_LOGFILE,ALL_ROLES).

     The following example is a LOG_ARCHIVE_DEST_*n* setting at the real-time capture downstream database:

     ```
     LOG_ARCHIVE_DEST_2='LOCATION=/home/arc_dest/srl_dbs1 MANDATORY
        VALID_FOR=(STANDBY_LOGFILE,PRIMARY_ROLE)'
     ```

     You can specify other attributes in the LOG_ARCHIVE_DEST_*n* initialization parameter if necessary.
   - Make sure the setting for the LOG_ARCHIVE_CONFIG initialization parameter includes the receive value.
   - Optionally set the LOG_ARCHIVE_FORMAT initialization parameter to generate the filenames in a specified format for the archived redo log files. The following example is a valid LOG_ARCHIVE_FORMAT setting:

     ```
     LOG_ARCHIVE_FORMAT=log%t_%s_%r.arc
     ```

   - Set the LOG_ARCHIVE_DEST_STATE_*n* initialization parameter that corresponds with the LOG_ARCHIVE_DEST_*n* parameter for the downstream database to ENABLE.

     For example, if the LOG_ARCHIVE_DEST_2 initialization parameter is set for the downstream database, then set one LOG_ARCHIVE_DEST_STATE_2 parameter in the following way:

     ```
     LOG_ARCHIVE_DEST_STATE_2=ENABLE
     ```

   - If you set other archive destinations at the downstream database, then, to keep archived standby redo log files separate from archived online redo log files from the downstream database, explicitly specify ONLINE_LOGFILE or STANDBY_LOGFILE, instead of ALL_LOGFILES, in the VALID_FOR attribute. For example, if the LOG_ARCHIVE_DEST_1 parameter specifies the archive destination for the online redo log files at the downstream database, then avoid the ALL_LOGFILES keyword in the VALID_FOR attribute when you set the LOG_ARCHIVE_DEST_1 parameter.

See Also: *Oracle Database Reference* and *Oracle Data Guard Concepts and Administration* for more information about these initialization parameters

5. If you reset any initialization parameters while an instance was running at a database in Step 3 or 4, then you might want to reset them in the relevant initialization parameter file as well, so that the new values are retained when the database is restarted.

   If you did not reset the initialization parameters while an instance was running, but instead reset them in the initialization parameter file in Step 3 or 4, then restart the database. The source database must be open when it sends redo data to the downstream database, because the global name of the source database is sent to the downstream database only if the source database is open.

6. At the downstream database, connect as an administrative user and create standby redo log files.

   Note: The following steps outline the general procedure for adding standby redo log files to the downstream database. The specific steps and SQL statements used to add standby redo log files depend on your environment. For example, in a Real Application Clusters environment, the steps are different. See *Oracle Data Guard Concepts and Administration* for detailed instructions about adding standby redo log files to a database.

   a. Determine the log file size used on the source database. The standby log file size must exactly match (or be larger than) the source database log file size. For example, if the source database log file size is 500 MB, then the standby log file size must be 500 MB or larger. You can determine the size of the redo log files at the source database (in bytes) by querying the `V$LOG` view at the source database.

   b. Determine the number of standby log file groups required on the downstream database. The number of standby log file groups must be at least one more than the number of online log file groups on the source database. For example, if the source database has two online log file groups, then the downstream database must have at least three standby log file groups. You can determine the number of source database online log file groups by querying the `V$LOG` view at the source database.

   c. Use the SQL statement `ALTER DATABASE ADD STANDBY LOGFILE` to add the standby log file groups to the downstream database.

      For example, assume that the source database has two online redo log file groups and is using a log file size of 500 MB. In this case, use the following statements to create the appropriate standby log file 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;
```

d. Ensure that the standby log file groups were added successfully by running the following query:

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

You 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
```

Creating a Real-Time Downstream Capture Process

This section contains an example that runs the CREATE_CAPTURE procedure in the DBMS_CAPTURE_ADM package to create a real-time downstream capture process at the dbs2.net downstream database that captures changes made to the dbs1.net source database. The capture process in this example uses a database link to dbs1.net for administrative purposes.

Complete the following steps:

1. Connect to the downstream database dbs2.net as the Streams administrator.

```
CONNECT strmadmin/strmadminpw@dbs2.net
```

2. Create the database link from dbs2.net to dbs1.net. For example, if the user strmadmin is the Streams administrator on both databases, then create the following database link:

```
CREATE DATABASE LINK dbs1.net CONNECT TO strmadmin IDENTIFIED BY strmadminpw
   USING 'dbs1.net';
```

This example assumes that a Streams administrator exists at the source database dbs1.net. If no Streams administrator exists at the source database, then the Streams administrator at the downstream database should connect to a user who allows remote access by a Streams administrator. You can enable remote access for a user by specifying the user as the grantee when you run the GRANT_REMOTE_ADMIN_ACCESS procedure in the DBMS_STREAMS_AUTH package at the source database.

3. Run the CREATE_CAPTURE procedure to create the capture process:

```
BEGIN
  DBMS_CAPTURE_ADM.CREATE_CAPTURE(
    queue_name         => 'strmadmin.streams_queue',
    capture_name       => 'real_time_capture',
    rule_set_name      => NULL,
    start_scn          => NULL,
    source_database    => 'dbs1.net',
    use_database_link  => true,
    first_scn          => NULL,
    logfile_assignment => 'implicit');
END;
/
```

Running this procedure performs the following actions:

- Creates a capture process named `real_time_capture` at the downstream database `dbs2.net`. A capture process with the same name must not exist.
- Associates the capture process with an existing **queue** on `dbs2.net` named `streams_queue`.
- Specifies that the source database of the changes that the capture process will capture is `dbs1.net`.
- Specifies that the capture process uses a database link with the same name as the source database global name to perform administrative actions at the source database.
- Specifies that the capture process accepts redo data implicitly from `dbs1.net`. Therefore, the capture process scans the standby redo log at `dbs2.net` for changes that it must capture. If the capture process falls behind, then it scans the archived redo log files written from the standby redo log.

This step does not associate the capture process `real_time_capture` with any **rule set**. A rule set will be created and associated with the capture process in the next step.

If no other capture process at `dbs2.net` is capturing changes from the `dbs1.net` source database, then the `DBMS_CAPTURE_ADM.BUILD` procedure is run automatically at `dbs1.net` using the database link. Running this procedure extracts the data dictionary at `dbs1.net` to the redo log, and a **LogMiner data dictionary** for `dbs1.net` is created at `dbs2.net` when the capture process `real_time_capture` is started for the first time at `dbs2.net`.

If multiple capture processes at `dbs2.net` are capturing changes from the `dbs1.net` source database, then the new capture process `real_time_capture` uses the same LogMiner data dictionary for `dbs1.net` as one of the existing archived-log capture process. Streams automatically chooses which LogMiner data dictionary to share with the new capture process.

Note: Only one real-time downstream capture process is allowed at a single downstream database.

See Also: "SCN Values Relating to a Capture Process" on page 2-19

4. Set the `downstream_real_time_mine` capture process parameter to `y`:

```
BEGIN
  DBMS_CAPTURE_ADM.SET_PARAMETER(
    capture_name => 'real_time_capture',
    parameter    => 'downstream_real_time_mine',
    value        => 'y');
END;
/
```

5. Create the **positive rule set** for the capture process and add a **rule** to it:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name          =>  'hr.departments',
    streams_type        =>  'capture',
    streams_name        =>  'real_time_capture',
    queue_name          =>  'strmadmin.streams_queue',
    include_dml         =>  true,
    include_ddl         =>  false,
    include_tagged_lcr  =>  false,
    source_database     =>  'dbs1.net',
    inclusion_rule      =>  true);
END;
/
```

   Running this procedure performs the following actions:

   - Creates a rule set at `dbs2.net` for capture process `real_time_capture`. The rule set has a system-generated name. The rule set is the positive rule set for the capture process because the `inclusion_rule` parameter is set to `true`.
   - Creates a rule that captures DML changes to the `hr.departments` table, and adds the rule to the positive rule set for the capture process. The rule has a system-generated name. The rule is added to the positive rule set for the capture process because the `inclusion_rule` parameter is set to `true`.
   - Prepares the `hr.departments` table at `dbs1.net` for **instantiation** using the database link created in Step 2.
   - Enables **supplemental logging** for any primary key, unique key, bitmap index, and foreign key columns in the `hr.departments` table. Primary key supplemental logging is required for the `hr.departments` table because this example creates a capture processes that captures changes to this table.

6. Connect to the source database `dbs1.net` as an administrative user with the necessary privileges to switch log files.

7. Archive the current log file at the source database:

```
ALTER SYSTEM ARCHIVE LOG CURRENT;
```

   Archiving the current log file at the source database starts real time mining of the source database redo log.

Now you can configure propagation or apply, or both, of the LCRs captured by the capture process.

In this example, if you want to use an **apply process** to apply the LCRs at the downstream database `dbs2.net`, then set the **instantiation SCN** for the `hr.departments` table at `dbs2.net`. If this table does not exist at `dbs2.net`, then instantiate it at `dbs2.net`.

For example, if the `hr.departments` table exists at `dbs2.net`, then set the instantiation SCN for the `hr.departments` table at `dbs2.net` by running the following procedure at the **destination database** `dbs2.net`:

```
DECLARE
  iscn  NUMBER;         -- Variable to hold instantiation SCN value
BEGIN
  iscn := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER@DBS1.NET;
  DBMS_APPLY_ADM.SET_TABLE_INSTANTIATION_SCN(
    source_object_name    => 'hr.departments',
    source_database_name  => 'dbs1.net',
    instantiation_scn     => iscn);
END;
/
```

After the instantiation SCN has been set, you can configure an apply process to apply LCRs for the `hr.departments` table from the `streams_queue` queue. Setting the instantiation SCN for an object at a database is required only if an apply process applies LCRs for the object. When all of the necessary propagations and apply processes are configured, start the capture process using the `START_CAPTURE` procedure in `DBMS_CAPTURE_ADM`.

Note: If you want the database objects to be synchronized at the source database and the destination database, then make sure the database objects are consistent when you set the instantiation SCN at the destination database. In the previous example, the `hr.departments` table should be consistent at the source and destination databases when the instantiation SCN is set.

See Also: *Oracle Streams Replication Administrator's Guide* for more information about instantiation

Creating an Archived-Log Downstream Capture Process that Assigns Logs Implicitly

To create a capture process that performs downstream capture, you must use the `CREATE_CAPTURE` procedure. The example in this section describes creating an **archived-log downstream capture process** that uses a database link to the **source database** for administrative purposes.

This example assumes the following:

- The source database is `dbs1.net` and the **downstream database** is `dbs2.net`.
- The capture process that will be created at `dbs2.net` uses the `streams_queue`.
- The capture process will capture DML changes to the `hr.departments` table.
- The capture process assigns log files implicitly. That is, the downstream capture process automatically scans all redo log files added by redo transport services or manually from the source database to the downstream database.

Preparing to Copy Redo Log Files for Archived-Log Downstream Capture

Whether a database link from the downstream database to the source database is used or not, complete the following steps to prepare the source database to copy its redo log files to the downstream database, and to prepare the downstream database to accept these redo log files:

1. Complete the tasks in "Preparing to Create a Capture Process" on page 11-3.
2. Configure Oracle Net so that the source database can communicate with the downstream database.

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

3. Set the following initialization parameters to configure redo transport services to copy archived redo log files from the source database to the downstream database:

   - At the source database, set at least one archive log destination in the LOG_ARCHIVE_DEST_*n* initialization parameter to a directory on the computer system running the downstream database. To do this, set the following attributes of this parameter:

     – SERVICE - Specify the network service name of the downstream database.

     – ARCH or LGWR ASYNC - If you specify ARCH (the default), then the archiver process (ARC*n*) will archive the redo log files to the downstream database. If you specify LGWR ASYNC, then the log writer process (LGWR) will archive the redo log files to the downstream database. Either ARCH or LGWR ASYNC is acceptable for a downstream database destination.

     – MANDATORY or OPTIONAL - If you specify MANDATORY, then archiving of a redo log file to the downstream database must succeed before the corresponding online redo log at the source database can be overwritten. If you specify OPTIONAL, then successful archiving of a redo log file to the downstream database is not required before the corresponding online redo log at the source database can be overwritten. Either MANDATORY or OPTIONAL is acceptable for a downstream database destination. If neither the MANDATORY nor the OPTIONAL attribute is specified, then the default is OPTIONAL.

     – NOREGISTER - Specify this attribute so that the downstream database location is not recorded in the downstream database control file.

     – TEMPLATE - Specify a directory and format template for archived redo logs at the downstream database. The TEMPLATE attribute overrides the LOG_ARCHIVE_FORMAT initialization parameter settings at the downstream database. The TEMPLATE attribute is valid only with remote destinations. Make sure the format uses all of the following variables at each source database: %t, %s, and %r.

   The following example is a LOG_ARCHIVE_DEST_*n* setting that specifies a downstream database:

```
LOG_ARCHIVE_DEST_2='SERVICE=DBS2.NET ARCH OPTIONAL NOREGISTER
   TEMPLATE=/usr/oracle/log_for_dbs1/dbs1_arch_%t_%s_%r.log'
```

   If another source database transfers log files to this downstream database, then, in the initialization parameter file at this other source database, you can use the TEMPLATE attribute to specify a different directory and format for the log files at the downstream database. The log files from each source database are kept separate at the downstream database.

   Tip: Log files from a remote source database should be kept separate from local database log files. In addition, if the downstream database contains log files from multiple source databases, then the log files from each source database should be kept separate from each other.

- At the source database, set the LOG_ARCHIVE_DEST_STATE_*n* initialization parameter that corresponds with the LOG_ARCHIVE_DEST_*n* parameter for the downstream database to ENABLE.

  For example, if the LOG_ARCHIVE_DEST_2 initialization parameter is set for the downstream database, then set one LOG_ARCHIVE_DEST_STATE_2 parameter in the following way:

  ```
  LOG_ARCHIVE_DEST_STATE_2=ENABLE
  ```

- At the source database, make sure the setting for the LOG_ARCHIVE_CONFIG initialization parameter includes the send value.
- At the downstream database, make sure the setting for the LOG_ARCHIVE_CONFIG initialization parameter includes the receive value.

  See Also: *Oracle Database Reference* and *Oracle Data Guard Concepts and Administration* for more information about these initialization parameters

4. If you reset any initialization parameters while the instance is running at a database in Step 3, then you might want to reset them in the initialization parameter file as well, so that the new values are retained when the database is restarted.

   If you did not reset the initialization parameters while the instance was running, but instead reset them in the initialization parameter file in Step 3, then restart the database. The source database must be open when it sends redo log files to the downstream database, because the global name of the source database is sent to the downstream database only if the source database is open.

Creating an Archived-Log Downstream Capture Process

This section contains an example that runs the CREATE_CAPTURE procedure in the DBMS_CAPTURE_ADM package to create an archived-log downstream capture process at the dbs2.net downstream database that captures changes made to the dbs1.net source database. The capture process in this example uses a database link to dbs1.net for administrative purposes.

Complete the following steps:

1. Connect to the downstream database dbs2.net as the Streams administrator.

   ```
   CONNECT strmadmin/strmadminpw@dbs2.net
   ```

2. Create the database link from dbs2.net to dbs1.net. For example, if the user strmadmin is the Streams administrator on both databases, then create the following database link:

   ```
   CREATE DATABASE LINK dbs1.net CONNECT TO strmadmin IDENTIFIED BY strmadminpw
      USING 'dbs1.net';
   ```

   This example assumes that a Streams administrator exists at the source database dbs1.net. If no Streams administrator exists at the source database, then the Streams administrator at the downstream database should connect to a user who allows remote access by a Streams administrator. You can enable remote access for a user by specifying the user as the grantee when you run the GRANT_REMOTE_ADMIN_ACCESS procedure in the DBMS_STREAMS_AUTH package at the source database.

3. While connected to the downstream database as the Streams administrator, run the CREATE_CAPTURE procedure to create the capture process:

```
BEGIN
  DBMS_CAPTURE_ADM.CREATE_CAPTURE(
    queue_name          => 'strmadmin.streams_queue',
    capture_name        => 'strm04_capture',
    rule_set_name       => NULL,
    start_scn           => NULL,
    source_database     => 'dbs1.net',
    use_database_link   => true,
    first_scn           => NULL,
    logfile_assignment => 'implicit');
END;
/
```

Running this procedure performs the following actions:

- Creates a capture process named strm04_capture at the downstream database dbs2.net. A capture process with the same name must not exist.
- Associates the capture process with an existing **queue** on dbs2.net named streams_queue.
- Specifies that the source database of the changes that the capture process will capture is dbs1.net.
- Specifies that the capture process accepts new redo log files implicitly from dbs1.net. Therefore, the capture process scans any new log files copied from dbs1.net to dbs2.net for changes that it must capture. These log files must be added to the capture process automatically using redo transport services or manually using the following DDL statement:

  ```
  ALTER DATABASE REGISTER LOGICAL LOGFILE file_name
     FOR capture_process;
  ```

  Here, *file_name* is the name of the redo log file and *capture_process* is the name of the capture process that will use the redo log file at the downstream database. You must add redo log files manually only if the logfile_assignment parameter is set to explicit.

This step does not associate the capture process strm04_capture with any **rule set**. A rule set will be created and associated with the capture process in the next step.

If no other capture process at dbs2.net is capturing changes from the dbs1.net source database, then the DBMS_CAPTURE_ADM.BUILD procedure is run automatically at dbs1.net using the database link. Running this procedure extracts the data dictionary at dbs1.net to the redo log, and a **LogMiner data dictionary** for dbs1.net is created at dbs2.net when the capture process is started for the first time at dbs2.net.

If multiple capture processes at dbs2.net are capturing changes from the dbs1.net source database, then the new capture process uses the same LogMiner data dictionary for dbs1.net as one of the existing capture process. Streams automatically chooses which LogMiner data dictionary to share with the new capture process.

See Also:

- "Capture Process Creation" on page 2-27
- *Oracle Database SQL Reference* for more information about the `ALTER DATABASE` statement
- *Oracle Data Guard Concepts and Administration* for more information registering redo log files

4. While connected to the downstream database as the Streams administrator, create the **positive rule set** for the capture process and add a **rule** to it:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name          =>  'hr.departments',
    streams_type        =>  'capture',
    streams_name        =>  'strm04_capture',
    queue_name          =>  'strmadmin.streams_queue',
    include_dml         =>  true,
    include_ddl         =>  false,
    include_tagged_lcr  =>  false,
    source_database     =>  'dbs1.net',
    inclusion_rule      =>  true);
END;
/
```

   Running this procedure performs the following actions:

   - Creates a rule set at `dbs2.net` for capture process `strm04_capture`. The rule set has a system-generated name. The rule set is a positive rule set for the capture process because the `inclusion_rule` parameter is set to `true`.
   - Creates a rule that captures DML changes to the `hr.departments` table, and adds the rule to the rule set for the capture process. The rule has a system-generated name. The rule is added to the positive rule set for the capture process because the `inclusion_rule` parameter is set to `true`.

Now you can configure propagation or apply, or both, of the LCRs captured by the `strm04_capture` capture process.

In this example, if you want to use an **apply process** to apply the LCRs at the downstream database `dbs2.net`, then set the **instantiation SCN** for the `hr.departments` table at `dbs2.net`. If this table does not exist at `dbs2.net`, then instantiate it at `dbs2.net`.

For example, if the `hr.departments` table exists at `dbs2.net`, then connect to the source database as the Streams administrator, and create a database link to `dbs2.net`:

```
CONNECT strmadmin/strmadminpw@dbs1.net

CREATE DATABASE LINK dbs2.net CONNECT TO strmadmin IDENTIFIED BY strmadminpw
   USING 'dbs2.net';
```

Set the instantiation SCN for the `hr.departments` table at `dbs2.net` by running the following procedure at the source database `dbs1.net`:

```
DECLARE
  iscn  NUMBER;         -- Variable to hold instantiation SCN value
BEGIN
  iscn := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER();
  DBMS_APPLY_ADM.SET_TABLE_INSTANTIATION_SCN@DBS2.NET(
    source_object_name    => 'hr.departments',
    source_database_name  => 'dbs1.net',
    instantiation_scn     => iscn);
END;
/
```

After the instantiation SCN has been set, you can configure an apply process to apply LCRs for the `hr.departments` table from the `streams_queue` queue. Setting the instantiation SCN for an object at a database is required only if an apply process applies LCRs for the object. When all of the necessary propagations and apply processes are configured, start the capture process using the `START_CAPTURE` procedure in `DBMS_CAPTURE_ADM`.

Note: If you want the database objects to be synchronized at the source database and the destination database, then make sure the database objects are consistent when you set the instantiation SCN at the destination database. In the previous example, the `hr.departments` table should be consistent at the source and destination databases when the instantiation SCN is set.

See Also: *Oracle Streams Replication Administrator's Guide* for more information about instantiation

Creating an Archived-Log Downstream Capture Process that Assigns Logs Explicitly

To create a capture process that performs downstream capture, you must use the `CREATE_CAPTURE` procedure. This section describes creating an **archived-log downstream capture process** that assigns redo log files explicitly. That is, you must use the `DBMS_FILE_TRANSFER` package, FTP, or some other method to transfer redo log files from the **source database** to the **downstream database**, and then you must register these redo log files with the downstream capture process manually.

In this example, assume the following:

- The source database is `dbs1.net` and the downstream database is `dbs2.net`.
- The capture process that will be created at `dbs2.net` uses the `streams_queue`.
- The capture process will capture DML changes to the `hr.departments` table.
- The capture process does not use a database link to the source database for administrative actions.

Complete the following steps:

1. Complete the tasks in "Preparing to Create a Capture Process" on page 11-3.
2. Connect to the source database `dbs1.net` as the Streams administrator. For example, if the Streams administrator is `strmadmin`, then issue the following statement:

   ```
   CONNECT strmadmin/strmadminpw@dbs1.net
   ```

   If you do not use a database link from the downstream database to the source database, then a Streams administrator must exist at the source database.

3. If there is no capture process at dbs2.net that captures changes from dbs1.net, then perform a build of the dbs1.net data dictionary in the redo log. This step is optional if a capture process at dbs2.net is already configured to capture changes from the dbs1.net source database.

```
SET SERVEROUTPUT ON
DECLARE
  scn  NUMBER;
BEGIN
  DBMS_CAPTURE_ADM.BUILD(
    first_scn => scn);
  DBMS_OUTPUT.PUT_LINE('First SCN Value = ' || scn);
END;
/
First SCN Value = 409391
```

   This procedure displays the valid **first SCN** value for the capture process that will be created at dbs2.net. Make a note of the SCN value returned because you will use it when you create the capture process at dbs2.net.

   If you run this procedure to build the data dictionary in the redo log, then when the capture process is first started at dbs2.net, it will create a **LogMiner data dictionary** using the data dictionary information in the redo log.

4. Prepare the hr.departments table for **instantiation**:

```
BEGIN
  DBMS_CAPTURE_ADM.PREPARE_TABLE_INSTANTIATION(
     table_name           =>  'hr.departments',
     supplemental_logging =>  'keys');
END;
/
```

   Primary key **supplemental logging** is required for the hr.departments table because this example creates a capture processes that captures changes to this table. Specifying keys for the supplemental_logging parameter in the PREPARE_TABLE_INSTANTIATION procedure enables supplemental logging for any primary key, unique key, bitmap index, and foreign key columns in the table.

5. Determine the current SCN of the source database:

```
SET SERVEROUTPUT ON SIZE 1000000

DECLARE
  iscn  NUMBER;         -- Variable to hold instantiation SCN value
BEGIN
  iscn := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER();
  DBMS_OUTPUT.PUT_LINE('Current SCN: ' || iscn);
END;
/
```

   You can use the returned SCN as the **instantiation SCN** for **destination database**s that will apply changes to the hr.departments table that were captured by the capture process being created. In this example, assume the returned SCN is 1001656.

6. Connect to the downstream database dbs2.net as the Streams administrator. For example, if the Streams administrator is strmadmin, then issue the following statement:

```
CONNECT strmadmin/strmadminpw@dbs2.net
```

7. Run the CREATE_CAPTURE procedure to create the capture process and specify the value obtained in Step 3 for the first_scn parameter:

```
BEGIN
  DBMS_CAPTURE_ADM.CREATE_CAPTURE(
    queue_name          => 'strmadmin.streams_queue',
    capture_name        => 'strm05_capture',
    rule_set_name       => NULL,
    start_scn           => NULL,
    source_database     => 'dbs1.net',
    use_database_link   => false,
    first_scn           => 409391, -- Use value from Step 3
    logfile_assignment => 'explicit');
END;
/
```

Running this procedure performs the following actions:

- Creates a capture process named strm05_capture at the downstream database dbs2.net. A capture process with the same name must not exist.
- Associates the capture process with an existing **queue** on dbs2.net named streams_queue.
- Specifies that the source database of the changes that the capture process will capture is dbs1.net.
- Specifies that the first SCN for the capture process is 409391. This value was obtained in Step 3. The first SCN is the lowest SCN for which a capture process can capture changes. Because a first SCN is specified, the capture process creates a new LogMiner data dictionary when it is first started, regardless of whether there are existing LogMiner data dictionaries for the same source database.
- Specifies new redo log files from dbs1.net must be assigned to the capture process explicitly. After a redo log file has been transferred to the computer running the downstream database, you assign the log file to the capture process explicitly using the following DDL statement:

  ```
  ALTER DATABASE REGISTER LOGICAL LOGFILE file_name FOR capture_process;
  ```

  Here, *file_name* is the name of the redo log file and *capture_process* is the name of the capture process that will use the redo log file at the downstream database. You must add redo log files manually if the logfile_assignment parameter is set to explicit.

This step does not associate the capture process strm05_capture with any **rule set**. A rule set will be created and associated with the capture process in the next step.

See Also:

- "Capture Process Creation" on page 2-27
- "SCN Values Relating to a Capture Process" on page 2-19
- *Oracle Database SQL Reference* for more information about the ALTER DATABASE statement
- *Oracle Data Guard Concepts and Administration* for more information registering redo log files

8. Create the **positive rule set** for the capture process and add a **rule** to it:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name          =>  'hr.departments',
    streams_type        =>  'capture',
    streams_name        =>  'strm05_capture',
    queue_name          =>  'strmadmin.streams_queue',
    include_dml         =>  true,
    include_ddl         =>  false,
    include_tagged_lcr  =>  false,
    source_database     =>  'dbs1.net',
    inclusion_rule      =>  true);
END;
/
```

   Running this procedure performs the following actions:

   - Creates a rule set at `dbs2.net` for capture process `strm05_capture`. The rule set has a system-generated name. The rule set is a positive rule set for the capture process because the `inclusion_rule` parameter is set to `true`.
   - Creates a rule that captures DML changes to the `hr.departments` table, and adds the rule to the rule set for the capture process. The rule has a system-generated name. The rule is added to the positive rule set for the capture process because the `inclusion_rule` parameter is set to `true`.

9. After the redo log file at the source database `dbs1.net` that contains the first SCN for the downstream capture process is archived, transfer the archived redo log file to the computer running the downstream database. The `BUILD` procedure in Step 3 determined the first SCN for the downstream capture process. If the redo log file is not yet archived, you can run the `ALTER SYSTEM SWITCH LOGFILE` statement on the database to archive it.

   You can run the following query at `dbs1.net` to identify the archived redo log file that contains the first SCN for the downstream capture process:

```
COLUMN NAME HEADING 'Archived Redo Log|File Name' FORMAT A50
COLUMN FIRST_CHANGE# HEADING 'First SCN' FORMAT 999999999

SELECT NAME, FIRST_CHANGE# FROM V$ARCHIVED_LOG
  WHERE FIRST_CHANGE# IS NOT NULL AND DICTIONARY_BEGIN = 'YES';
```

   Transfer the archived redo log file with a `FIRST_CHANGE#` that matches the first SCN returned in Step 3 to the computer running the downstream capture process.

10. At the downstream database `dbs2.net`, connect as an administrative user and assign the transferred redo log file to the capture process. For example, if the redo log file is `/oracle/logs_from_dbs1/1_10_486574859.dbf`, then issue the following statement:

```
ALTER DATABASE REGISTER LOGICAL LOGFILE
   '/oracle/logs_from_dbs1/1_10_486574859.dbf' FOR 'strm05_capture';
```

Now you can configure propagation or apply, or both, of the LCRs captured by the `strm05_capture` capture process.

In this example, if you want to use an **apply process** to apply the LCRs at the downstream database `dbs2.net`, then set the instantiation SCN for the `hr.departments` table at `dbs2.net`. If this table does not exist at `dbs2.net`, then instantiate it at `dbs2.net`.

For example, if the hr.departments table exists at dbs2.net, then set the instantiation SCN for the hr.departments table at dbs2.net to the value determined in Step 5. Run the following procedure at dbs2.net to set the instantiation SCN for the hr.departments table:

```
CONNECT strmadmin/strmadminpw@dbs2.net

BEGIN
  DBMS_APPLY_ADM.SET_TABLE_INSTANTIATION_SCN(
    source_object_name    => 'hr.departments',
    source_database_name  => 'dbs1.net',
    instantiation_scn     => 1001656);
END;
/
```

After the instantiation SCN has been set, you can configure an apply process to apply LCRs for the hr.departments table from the streams_queue queue. Setting the instantiation SCN for an object at a database is required only if an apply process applies LCRs for the object. When all of the necessary propagations and apply processes are configured, start the capture process using the START_CAPTURE procedure in DBMS_CAPTURE_ADM.

Note: If you want the database objects to be synchronized at the source database and the destination database, then make sure the database objects are consistent when you set the instantiation SCN at the destination database. In the previous example, the hr.departments table should be consistent at the source and destination databases when the instantiation SCN is set.

See Also: *Oracle Streams Replication Administrator's Guide* for more information about instantiation

Creating a Local Capture Process with Non-NULL Start SCN

This example runs the CREATE_CAPTURE procedure in the DBMS_CAPTURE_ADM package to create a **local capture process** with a **start SCN** set to 223525. This example assumes that there is at least one local capture process at the database, and that this capture process has taken at least one **checkpoint**. You can always specify a start SCN for a new capture process that is equal to or greater than the current SCN of the **source database**. If you want to specify a start SCN that is lower than the current SCN of the database, then the specified start SCN must be higher than the lowest **first SCN** for an existing local capture process that has been started successfully at least once and has taken at least one checkpoint.

You can determine the first SCN for existing capture processes, and whether these capture processes have taken a checkpoint, by running the following query:

```
SELECT CAPTURE_NAME, FIRST_SCN, MAX_CHECKPOINT_SCN FROM DBA_CAPTURE;
```

Your output looks similar to the following:

```
CAPTURE_NAME                   FIRST_SCN MAX_CHECKPOINT_SCN
------------------------------ ---------- ------------------
CAPTURE_SIMP                      223522             230825
```

These results show that the capture_simp capture process has a first SCN of 223522. Also, this capture process has taken a checkpoint because the MAX_CHECKPOINT_SCN value is non-NULL. Therefore, the start SCN for the new capture process can be set to 223522 or higher.

Before you proceed, complete the tasks in "Preparing to Create a Capture Process" on page 11-3. Next, run the following procedure to create the capture process:

```
BEGIN
  DBMS_CAPTURE_ADM.CREATE_CAPTURE(
    queue_name          => 'strmadmin.streams_queue',
    capture_name        => 'strm05_capture',
    rule_set_name       => 'strmadmin.strm01_rule_set',
    start_scn           => 223525,
    source_database     => NULL,
    use_database_link   => false,
    first_scn           => NULL);
END;
/
```

Running this procedure performs the following actions:

- Creates a capture process named strm05_capture. A capture process with the same name must not exist.
- Associates the capture process with an existing **queue** named streams_queue.
- Associates the capture process with an existing **rule set** named strm01_rule_set. This rule set is the **positive rule set** for the capture process.
- Specifies 223525 as the start SCN for the capture process. The new capture process uses the same **LogMiner data dictionary** as one of the existing capture processes. Streams automatically chooses which LogMiner data dictionary to share with the new capture process. Because the first_scn parameter was set to NULL, the first SCN for the new capture process is the same as the first SCN of the existing capture process whose LogMiner data dictionary was shared. In this example, the existing capture process is capture_simp.
- Creates a capture process that captures local changes to the source database because the source_database parameter is set to NULL. For a local capture process, you can also specify the global name of the local database for this parameter.

Note: If no local capture process exists when the procedure in this example is run, then the DBMS_CAPTURE_ADM.BUILD procedure is run automatically during capture process creation to extract the data dictionary into the redo log. The first time the new capture process is started, it uses this redo data to create a LogMiner data dictionary. In this case, a specified start_scn parameter value must be equal to or higher than the current database SCN.

See Also:

- "Capture Process Creation" on page 2-27
- "First SCN and Start SCN Specifications During Capture Process Creation" on page 2-33

Starting a Capture Process

You run the START_CAPTURE procedure in the DBMS_CAPTURE_ADM package to start an existing **capture process**. For example, the following procedure starts a capture process named strm01_capture:

```
BEGIN
  DBMS_CAPTURE_ADM.START_CAPTURE(
    capture_name => 'strm01_capture');
END;
/
```

Note: If a new capture process will use a new **LogMiner data dictionary**, then, when you first start the new capture process, some time might be required to populate the new LogMiner data dictionary. A new LogMiner data dictionary is created if a non-NULL first SCN value was specified when the capture process was created.

Stopping a Capture Process

You run the STOP_CAPTURE procedure in the DBMS_CAPTURE_ADM package to stop an existing **capture process**. For example, the following procedure stops a capture process named strm01_capture:

```
BEGIN
  DBMS_CAPTURE_ADM.STOP_CAPTURE(
    capture_name => 'strm01_capture');
END;
/
```

Managing the Rule Set for a Capture Process

This section contains instructions for completing the following tasks:

- Specifying a Rule Set for a Capture Process
- Adding Rules to a Rule Set for a Capture Process
- Removing a Rule from a Rule Set for a Capture Process
- Removing a Rule Set for a Capture Process

See Also:

- Chapter 5, "Rules"
- Chapter 6, "How Rules Are Used in Streams"

Specifying a Rule Set for a Capture Process

You can specify one **positive rule set** and one **negative rule set** for a **capture process**. The capture process captures a change if it evaluates to TRUE for at least one **rule** in the positive rule set and evaluates to FALSE for all the rules in the negative rule set. The negative rule set is evaluated before the positive rule set.

See Also:

- Chapter 5, "Rules"
- Chapter 6, "How Rules Are Used in Streams"

Specifying a Positive Rule Set for a Capture Process

You specify an existing **rule set** as the positive rule set for an existing capture process using the rule_set_name parameter in the ALTER_CAPTURE procedure. This procedure is in the DBMS_CAPTURE_ADM package.

For example, the following procedure sets the positive rule set for a capture process named strm01_capture to strm02_rule_set.

```
BEGIN
  DBMS_CAPTURE_ADM.ALTER_CAPTURE(
    capture_name  => 'strm01_capture',
    rule_set_name => 'strmadmin.strm02_rule_set');
END;
/
```

Specifying a Negative Rule Set for a Capture Process

You specify an existing rule set as the negative rule set for an existing capture process using the negative_rule_set_name parameter in the ALTER_CAPTURE procedure. This procedure is in the DBMS_CAPTURE_ADM package.

For example, the following procedure sets the negative rule set for a capture process named strm01_capture to strm03_rule_set.

```
BEGIN
  DBMS_CAPTURE_ADM.ALTER_CAPTURE(
    capture_name          => 'strm01_capture',
    negative_rule_set_name => 'strmadmin.strm03_rule_set');
END;
/
```

Adding Rules to a Rule Set for a Capture Process

To add **rule**s to a **rule set** for an existing **capture process**, you can run one of the following procedures in the DBMS_STREAMS_ADM package and specify the existing capture process:

- ADD_TABLE_RULES
- ADD_SUBSET_RULES
- ADD_SCHEMA_RULES
- ADD_GLOBAL_RULES

Excluding the ADD_SUBSET_RULES procedure, these procedures can add rules to the **positive rule set** or **negative rule set** for a capture process. The ADD_SUBSET_RULES procedure can add rules only to the positive rule set for a capture process.

See Also: "System-Created Rules" on page 6-5

Adding Rules to the Positive Rule Set for a Capture Process

The following example runs the ADD_TABLE_RULES procedure in the DBMS_STREAMS_ADM package to add rules to the positive rule set of a capture process named strm01_capture:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name      =>  'hr.departments',
    streams_type    =>  'capture',
    streams_name    =>  'strm01_capture',
    queue_name      =>  'strmadmin.streams_queue',
    include_dml     =>  true,
    include_ddl     =>  true,
    inclusion_rule  =>  true);
END;
/
```

Running this procedure performs the following actions:

- Creates two rules. One rule evaluates to TRUE for DML changes to the hr.departments table, and the other rule evaluates to TRUE for DDL changes to the hr.departments table. The rule names are system generated.
- Adds the two rules to the positive rule set associated with the capture process because the inclusion_rule parameter is set to true.
- Prepares the hr.departments table for **instantiation** by running the PREPARE_TABLE_INSTANTIATION procedure in the DBMS_CAPTURE_ADM package.
- Enables **supplemental logging** for any primary key, unique key, bitmap index, and foreign key columns in the hr.departments table. When the PREPARE_TABLE_INSTANTIATION procedure is run, the default value (keys) is specified for the supplemental_logging parameter.

If the capture process is performing downstream capture, then the table is prepared for instantiation and supplemental logging is enabled for key columns only if the **downstream capture process** uses a database link to the **source database**. If a downstream capture process does not use a database link to the source database, then the table must be prepared for instantiation manually and supplemental logging must be enabled manually.

Adding Rules to the Negative Rule Set for a Capture Process

The following example runs the ADD_TABLE_RULES procedure in the DBMS_STREAMS_ADM package to add rules to the **negative rule set** of a capture process named strm01_capture:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name      =>  'hr.job_history',
    streams_type    =>  'capture',
    streams_name    =>  'strm01_capture',
    queue_name      =>  'strmadmin.streams_queue',
    include_dml     =>  true,
    include_ddl     =>  true,
    inclusion_rule  =>  false);
END;
/
```

Running this procedure performs the following actions:

- Creates two rules. One rule evaluates to TRUE for DML changes to the hr.job_history table, and the other rule evaluates to TRUE for DDL changes to the hr.job_history table. The rule names are system generated.
- Adds the two rules to the negative rule set associated with the capture process, because the inclusion_rule parameter is set to false.

Removing a Rule from a Rule Set for a Capture Process

You specify that you want to remove a **rule** from a **rule set** for an existing **capture process** by running the REMOVE_RULE procedure in the DBMS_STREAMS_ADM package. For example, the following procedure removes a rule named departments3 from the **positive rule set** of a capture process named strm01_capture.

```
BEGIN
  DBMS_STREAMS_ADM.REMOVE_RULE(
    rule_name        => 'departments3',
    streams_type     => 'capture',
    streams_name     => 'strm01_capture',
    drop_unused_rule => true,
    inclusion_rule   => true);
END;
/
```

In this example, the drop_unused_rule parameter in the REMOVE_RULE procedure is set to true, which is the default setting. Therefore, if the rule being removed is not in any other rule set, then it will be dropped from the database. If the drop_unused_rule parameter is set to false, then the rule is removed from the rule set, but it is not dropped from the database.

If the inclusion_rule parameter is set to false, then the REMOVE_RULE procedure removes the rule from the **negative rule set** for the capture process, not the positive rule set.

If you want to remove all of the rules in a rule set for the capture process, then specify NULL for the rule_name parameter when you run the REMOVE_RULE procedure.

> **See Also:** "Streams Client with One or More Empty Rule Sets" on page 6-4

Removing a Rule Set for a Capture Process

You specify that you want to remove a **rule set** from an existing **capture process** using the ALTER_CAPTURE procedure in the DBMS_CAPTURE_ADM package. This procedure can remove the **positive rule set**, **negative rule set**, or both. Specify true for the remove_rule_set parameter to remove the positive rule set for the capture process. Specify true for the remove_negative_rule_set parameter to remove the negative rule set for the capture process.

For example, the following procedure removes both the positive and negative rule set from a capture process named strm01_capture.

```
BEGIN
  DBMS_CAPTURE_ADM.ALTER_CAPTURE(
    capture_name             => 'strm01_capture',
    remove_rule_set          => true,
    remove_negative_rule_set => true);
END;
/
```

Note: If a capture process does not have a positive or negative rule set, then the capture process captures all supported changes to all objects in the database, excluding database objects in the SYS, SYSTEM, and CTXSYS schemas.

Setting a Capture Process Parameter

Set a **capture process** parameter using the SET_PARAMETER procedure in the DBMS_CAPTURE_ADM package. Capture process parameters control the way a capture process operates.

For example, the following procedure sets the parallelism parameter for a capture process named strm01_capture to 3.

```
BEGIN
  DBMS_CAPTURE_ADM.SET_PARAMETER(
    capture_name => 'strm01_capture',
    parameter    => 'parallelism',
    value        => '3');
END;
/
```

Note:

- Setting the parallelism parameter automatically stops and restarts a capture process.
- The value parameter is always entered as a VARCHAR2 value, even if the parameter value is a number.

See Also:

- "Capture Process Architecture" on page 2-22
- The DBMS_CAPTURE_ADM.SET_PARAMETER procedure in the *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the capture process parameters

Setting the Capture User for a Capture Process

The **capture user** is the user who captures all DML changes and DDL changes that satisfy the **capture process rule set**s. Set the capture user for a capture process using the capture_user parameter in the ALTER_CAPTURE procedure in the DBMS_CAPTURE_ADM package.

To change the capture user, the user who invokes the ALTER_CAPTURE procedure must be granted DBA role. Only the SYS user can set the capture_user to SYS.

For example, the following procedure sets the capture user for a capture process named strm01_capture to hr.

```
BEGIN
  DBMS_CAPTURE_ADM.ALTER_CAPTURE(
    capture_name => 'strm01_capture',
    capture_user => 'hr');
END;
/
```

Running this procedure grants the new capture user enqueue privilege on the queue used by the capture process and configures the user as a **secure queue** user of the **queue**. In addition, make sure the capture user has the following privileges:

- EXECUTE privilege on the rule sets used by the capture process
- EXECUTE privilege on all **custom rule-based transformation** functions used in the rule set

These privileges must be granted directly to the capture user. They cannot be granted through roles.

Managing the Checkpoint Retention Time for a Capture Process

The **checkpoint retention time** is the amount of time that a **capture process** retains **checkpoint**s before purging them automatically. Set the checkpoint retention time for a capture process using checkpoint_retention_time parameter in the ALTER_CAPTURE procedure of the DBMS_CAPTURE_ADM package.

See Also: "Capture Process Checkpoints" on page 2-25

Setting the Checkpoint Retention Time for a Capture Process to a New Value

When you set the checkpoint retention time, you can specify partial days with decimal values. For example, run the following procedure to specify that a capture process named strm01_capture should purge checkpoints automatically every ten days and twelve hours:

```
BEGIN
  DBMS_CAPTURE_ADM.ALTER_CAPTURE(
    capture_name              => 'strm01_capture',
    checkpoint_retention_time => 10.5);
END;
/
```

Setting the Checkpoint Retention Time for a Capture Process to Infinite

To specify that a capture process should not purge checkpoints automatically, set the checkpoint retention time to DBMS_CAPTURE_ADM.INFINITE. For example, the following procedure sets the checkpoint retention time for a name strm01_capture to infinite:

```
BEGIN
  DBMS_CAPTURE_ADM.ALTER_CAPTURE(
    capture_name              => 'strm01_capture',
    checkpoint_retention_time => DBMS_CAPTURE_ADM.INFINITE);
END;
/
```

Specifying Supplemental Logging at a Source Database

Supplemental logging must be specified for some columns at a **source database** for changes to the columns to be applied successfully at a **destination database**. Typically, **supplemental logging** is required in Streams **replication** environments, but it might be required in any environment that processes **captured message**s with an **apply process**. You use the ALTER DATABASE statement to specify supplemental logging for all tables in a database, and you use the ALTER TABLE statement to specify supplemental logging for a particular table.

See Also: *Oracle Streams Replication Administrator's Guide* for more information about specifying supplemental logging

Adding an Archived Redo Log File to a Capture Process Explicitly

You can add an archived redo log file to a **capture process** manually using the following statement:

```
ALTER DATABASE REGISTER LOGICAL LOGFILE
   file_name FOR capture_process;
```

Here, *file_name* is the name of the archived redo log file being added, and *capture_process* is the name of the capture process that will use the redo log file at the **downstream database**. The *capture_process* is equivalent to the *logminer_session_name* and must be specified. The redo log file must be present at the site running capture process.

For example, to add the `/usr/log_files/1_3_486574859.dbf` archived redo log file to a capture process named `strm03_capture`, issue the following statement:

```
ALTER DATABASE REGISTER LOGICAL LOGFILE '/usr/log_files/1_3_486574859.dbf'
  FOR 'strm03_capture';
```

See Also: *Oracle Database SQL Reference* for more information about the `ALTER DATABASE` statement and *Oracle Data Guard Concepts and Administration* for more information registering redo log files

Setting the First SCN for an Existing Capture Process

You can set the **first SCN** for an existing **capture process** using the `ALTER_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package.

The specified first SCN must meet the following requirements:

- It must be greater than the current first SCN for the capture process.
- It must be less than or equal to the current **applied SCN** for the capture process. However, this requirement does not apply if the current applied SCN for the capture process is zero.
- It must be less than or equal to the **required checkpoint SCN** for the capture process.

You can determine the current first SCN, applied SCN, and required checkpoint SCN for each capture process in a database using the following query:

```
SELECT CAPTURE_NAME, FIRST_SCN, APPLIED_SCN, REQUIRED_CHECKPOINT_SCN
   FROM DBA_CAPTURE;
```

When you reset a first SCN for a capture process, information below the new first SCN setting is purged from the **LogMiner data dictionary** for the capture process automatically. Therefore, after the first SCN is reset for a capture process, the **start SCN** for the capture process cannot be set lower than the new first SCN. Also, redo log files that contain information prior to the new first SCN setting will never be needed by the capture process.

For example, the following procedure sets the first SCN for a capture process named `strm01_capture` to `351232`.

```
BEGIN
  DBMS_CAPTURE_ADM.ALTER_CAPTURE(
    capture_name => 'strm01_capture',
    first_scn    => 351232);
END;
/
```

Note:

- If the specified first SCN is higher than the current start SCN for the capture process, then the start SCN is set automatically to the new value of the first SCN.
- If you need to capture changes in the redo log from a point in time in the past, then you can create a new capture process and specify a first SCN that corresponds to a previous data dictionary build in the redo log. The `BUILD` procedure in the `DBMS_CAPTURE_ADM` package performs a data dictionary build in the redo log.
- You can query the `DBA_LOGMNR_PURGED_LOG` data dictionary view to determine which redo log files will never be needed by any capture process.

See Also:

- "SCN Values Relating to a Capture Process" on page 2-19
- "The LogMiner Data Dictionary for a Capture Process" on page 2-28
- "First SCN and Start SCN Specifications During Capture Process Creation" on page 2-33
- "Displaying SCN Values for Each Redo Log File Used by Each Capture Process" on page 20-9 for a query that determines which redo log files are no longer needed

Setting the Start SCN for an Existing Capture Process

You can set the **start SCN** for an existing **capture process** using the `ALTER_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package. Typically, you reset the start SCN for a capture process if point-in-time recovery must be performed on one of the **destination database**s that receive changes from the capture process.

The specified start SCN must be greater than or equal to the **first SCN** for the capture process. When you reset a start SCN for a capture process, make sure the required redo log files are available to the capture process.

You can determine the first SCN for each capture process in a database using the following query:

```
SELECT CAPTURE_NAME, FIRST_SCN FROM DBA_CAPTURE;
```

For example, the following procedure sets the start SCN for a capture process named `strm01_capture` to `750338`.

```
BEGIN
  DBMS_CAPTURE_ADM.ALTER_CAPTURE(
    capture_name => 'strm01_capture',
    start_scn    => 750338);
END;
/
```

See Also:

- "SCN Values Relating to a Capture Process" on page 2-19
- *Oracle Streams Replication Administrator's Guide* for information about performing database point-in-time recovery on a destination database in a Streams environment

Specifying Whether Downstream Capture Uses a Database Link

You specify whether an existing **downstream capture process** uses a database link to the **source database** for administrative purposes using the `ALTER_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package. Set the `use_database_link` parameter to `true` to specify that the downstream capture process uses a database link, or you set the `use_database_link` parameter to `false` to specify that the downstream capture process does not use a database link.

If you want a capture process that is not using a database link currently to begin using a database link, then specify `true` for the `use_database_link` parameter. In this case, a database link with the same name as the global name as the source database must exist at the **downstream database**.

If you want a capture process that is using a database link currently to stop using a database link, then specify `false` for the `use_database_link` parameter. In this case, some administration must be performed manually after you alter the capture process. For example, if you add new capture process **rule**s using the `DBMS_STREAMS_ADM` package, then you must prepare the objects relating to the rules for **instantiation** manually at the source database.

If you specify `NULL` for the `use_database_link` parameter, then the current value of this parameter for the capture process is not changed.

The example in "Creating an Archived-Log Downstream Capture Process that Assigns Logs Explicitly" on page 11-18 created the capture process `strm05_capture` and specified that this capture process does not use a database link. To create a database link to the source database `dbs1.net` and specify that this capture process uses the database link, complete the following actions:

```
CREATE DATABASE LINK dbs1.net CONNECT TO strmadmin IDENTIFIED BY strmadminpw
   USING 'dbs1.net';

BEGIN
  DBMS_CAPTURE_ADM.ALTER_CAPTURE(
    capture_name       => 'strm05_capture',
    use_database_link  => true);
END;
/
```

See Also: "Local Capture and Downstream Capture" on page 2-12

Managing Extra Attributes in Captured Messages

You can use the INCLUDE_EXTRA_ATTRIBUTE procedure in the DBMS_CAPTURE_ADM package to instruct a **capture process** to capture one or more extra attributes. You can also use this procedure to instruct a capture process to exclude an extra attribute that it is capturing currently.

The extra attributes are the following:

- row_id (row LCRs only)
- serial#
- session#
- thread#
- tx_name
- username

This section contains instructions for completing the following tasks:

- Including Extra Attributes in Captured Messages
- Excluding Extra Attributes from Captured Messages

See Also:

- "Extra Information in LCRs" on page 2-5
- "Viewing the Extra Attributes Captured by Each Capture Process" on page 20-12
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the INCLUDE_EXTRA_ATTRIBUTE procedure

Including Extra Attributes in Captured Messages

To instruct a capture process named strm01_capture to include the transaction name in each **captured message**, run the following procedure:

```
BEGIN
  DBMS_CAPTURE_ADM.INCLUDE_EXTRA_ATTRIBUTE(
    capture_name   => 'strm01_capture',
    attribute_name => 'tx_name',
    include        => true);
END;
/
```

Excluding Extra Attributes from Captured Messages

To instruct a capture process named strm01_capture to exclude the transaction name from each **captured message**, run the following procedure:

```
BEGIN
  DBMS_CAPTURE_ADM.INCLUDE_EXTRA_ATTRIBUTE(
    capture_name   => 'strm01_capture',
    attribute_name => 'tx_name',
    include        => false);
END;
/
```

Dropping a Capture Process

You run the DROP_CAPTURE procedure in the DBMS_CAPTURE_ADM package to drop an existing **capture process**. For example, the following procedure drops a capture process named strm02_capture:

```
BEGIN
  DBMS_CAPTURE_ADM.DROP_CAPTURE(
    capture_name          => 'strm02_capture',
    drop_unused_rule_sets => true);
END;
/
```

Because the drop_unused_rule_sets parameter is set to true, this procedure also drops any **rule set**s used by the strm02_capture capture process, unless a rule set is used by another **Streams client**. If the drop_unused_rule_sets parameter is set to true, then both the **positive rule set** and **negative rule set** for the capture process might be dropped. If this procedure drops a rule set, then it also drops any **rule**s in the rule set that are not in another rule set.

Note: The status of a capture process must be DISABLED or ABORTED before it can be dropped. You cannot drop an ENABLED capture process.

12

Managing Staging and Propagation

This chapter provides instructions for managing ANYDATA queues, **propagation**s, and messaging environments.

This chapter contains these topics:

- Managing ANYDATA Queues
- Managing Streams Propagations and Propagation Jobs
- Managing a Streams Messaging Environment

Each task described in this chapter should be completed by a Streams administrator that has been granted the appropriate privileges, unless specified otherwise.

See Also:

- Chapter 3, "Streams Staging and Propagation"
- "Configuring a Streams Administrator" on page 10-1

Managing ANYDATA Queues

An ANYDATA queue stages **message**s whose payloads are of ANYDATA type. Therefore, an ANYDATA queue can stage a message with a payload of nearly any type, if the payload is wrapped in an ANYDATA wrapper. Each Streams **capture process**, **apply process**, and **messaging client** is associated with one ANYDATA queue, and each Streams **propagation** is associated with one ANYDATA **source queue** and one ANYDATA **destination queue**.

This section contains instructions for completing the following tasks related to ANYDATA queues:

- Creating an ANYDATA Queue
- Enabling a User to Perform Operations on a Secure Queue
- Disabling a User from Performing Operations on a Secure Queue
- Removing an ANYDATA Queue

Creating an ANYDATA Queue

The easiest way to create an ANYDATA queue is to use the SET_UP_QUEUE procedure in the DBMS_STREAMS_ADM package. This procedure enables you to specify the following settings for the ANYDATA queue it creates:

- The **queue table** for the **queue**
- A storage clause for the queue table
- The queue name
- A queue user that will be configured as a **secure queue** user of the queue and granted ENQUEUE and DEQUEUE privileges on the queue
- A comment for the queue

If the specified queue table does not exist, then it is created. If the specified queue table exists, then the existing queue table is used for the new queue. If you do not specify any queue table when you create the queue, then, by default, streams_queue_table is specified.

You can use a single procedure, the SET_UP_QUEUE procedure in the DBMS_STREAMS_ADM package, to create an ANYDATA queue and the queue table used by the queue. For SET_UP_QUEUE to create a new queue table, the specified queue table must not exist.

For example, run the following procedure to create an ANYDATA queue with the SET_UP_QUEUE procedure:

```
BEGIN
  DBMS_STREAMS_ADM.SET_UP_QUEUE(
    queue_table => 'strmadmin.strm01_queue_table',
    queue_name  => 'strmadmin.strm01_queue',
    queue_user  => 'hr');
END;
/
```

Running this procedure performs the following actions:

- Creates a queue table named strm01_queue_table. The queue table is created only if it does not already exist. Queues based on the queue table stage messages of ANYDATA type. Queue table names can be a maximum of 24 bytes.
- Creates a queue named strm01_queue. The queue is created only if it does not already exist. Queue names can be a maximum of 24 bytes.
- Specifies that the strm01_queue queue is based on the strmadmin.strm01_queue_table queue table.
- Configures the hr user as a secure queue user of the queue, and grants this user ENQUEUE and DEQUEUE privileges on the queue.
- Starts the queue.

Default settings are used for the parameters that are not explicitly set in the SET_UP_QUEUE procedure.

When the SET_UP_QUEUE procedure creates a queue table, the following DBMS_AQADM.CREATE_QUEUE_TABLE parameter settings are specified:

- If the database is Oracle Database 10*g* Release 2 or later, the sort_list setting is commit_time. If the database is a release prior to Oracle Database 10*g* Release 2, the sort_list setting is enq_time.
- The multiple_consumers setting is true.
- The message_grouping setting is transactional.
- The secure setting is true.

The other parameters in the CREATE_QUEUE_TABLE procedure are set to their default values.

You can use the CREATE_QUEUE_TABLE procedure in the DBMS_AQADM package to create a queue table of ANYDATA type with different properties than the default properties specified by the SET_UP_QUEUE procedure in the DBMS_STREAMS_ADM package. After you create the queue table with the CREATE_QUEUE_TABLE procedure, you can create a queue that uses the queue table. To do so, specify the queue table in the queue_table parameter of the SET_UP_QUEUE procedure.

Similarly, you can use the CREATE_QUEUE procedure in the DBMS_AQADM package to create a queue instead of SET_UP_QUEUE. Use CREATE_QUEUE if you require custom settings for the queue. For example, use CREATE_QUEUE to specify a custom retry delay or retention time. If you use CREATE_QUEUE, then you must start the queue manually.

Note: A message cannot be enqueued into a queue unless a subscriber who can dequeue the message is configured.

See Also:

- "Wrapping User Message Payloads in an ANYDATA Wrapper and Enqueuing Them" on page 12-15 for an example that creates an ANYDATA queue using procedures in the DBMS_AQADM package
- "ANYDATA Queues and User Messages" on page 3-11
- "Commit-Time Queues" on page 3-14
- "Secure Queues" on page 3-23
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the SET_UP_QUEUE, CREATE_QUEUE_TABLE, and CREATE_QUEUE procedures

Enabling a User to Perform Operations on a Secure Queue

For a user to perform **queue** operations, such as enqueue and dequeue, on a **secure queue**, the user must be configured as a secure queue user of the queue. If you use the SET_UP_QUEUE procedure in the DBMS_STREAMS_ADM package to create the secure queue, then the queue owner and the user specified by the queue_user parameter are configured as secure users of the queue automatically. If you want to enable other users to perform operations on the queue, then you can configure these users in one of the following ways:

- Run SET_UP_QUEUE and specify a queue_user. Queue creation is skipped if the queue already exists, but a new queue user is configured if one is specified.
- Associate the user with an AQ agent manually.

The following example illustrates associating a user with an AQ agent manually. Suppose you want to enable the oe user to perform queue operations on the strm01_queue created in "Creating an ANYDATA Queue" on page 12-2. The following steps configure the oe user as a secure queue user of strm01_queue:

1. Connect as an administrative user who can create AQ agents and alter users.
2. Create an agent:

```
EXEC DBMS_AQADM.CREATE_AQ_AGENT(agent_name => 'strm01_queue_agent');
```

3. If the user must be able to dequeue **message**s from queue, then make the agent a subscriber of the secure queue:

```
DECLARE
  subscriber SYS.AQ$_AGENT;
BEGIN
  subscriber :=  SYS.AQ$_AGENT('strm01_queue_agent', NULL, NULL);
  DBMS_AQADM.ADD_SUBSCRIBER(
    queue_name          =>  'strmadmin.strm01_queue',
    subscriber          =>  subscriber,
    rule                =>  NULL,
    transformation      =>  NULL);
END;
/
```

4. Associate the user with the agent:

```
BEGIN
  DBMS_AQADM.ENABLE_DB_ACCESS(
    agent_name  => 'strm01_queue_agent',
    db_username => 'oe');
END;
/
```

5. Grant the user EXECUTE privilege on the DBMS_STREAMS_MESSAGING package or the DBMS_AQ package, if the user is not already granted these privileges:

```
GRANT EXECUTE ON DBMS_STREAMS_MESSAGING TO oe;

GRANT EXECUTE ON DBMS_AQ TO oe;
```

When these steps are complete, the oe user is a secure user of the strm01_queue queue and can perform operations on the queue. You still must grant the user specific privileges to perform queue operations, such as enqueue and dequeue privileges.

See Also:

- "Secure Queues" on page 3-23
- *Oracle Database PL/SQL Packages and Types Reference* for more information about AQ agents and using the DBMS_AQADM package

Disabling a User from Performing Operations on a Secure Queue

You might want to disable a user from performing **queue** operations on a **secure queue** for the following reasons:

- You dropped a **capture process**, but you did not drop the queue that was used by the capture process, and you do not want the user who was the **capture user** to be able to perform operations on the remaining secure queue.
- You dropped an **apply process**, but you did not drop the queue that was used by the apply process, and you do not want the user who was the **apply user** to be able to perform operations on the remaining secure queue.
- You used the `ALTER_APPLY` procedure in the `DBMS_APPLY_ADM` package to change the `apply_user` for an apply process, and you do not want the old `apply_user` to be able to perform operations on the apply process queue.
- You enabled a user to perform operations on a secure queue by completing the steps described in Enabling a User to Perform Operations on a Secure Queue on page 12-3, but you no longer want this user to be able to perform operations on the secure queue.

To disable a secure queue user, you can revoke `ENQUEUE` and `DEQUEUE` privilege on the queue from the user, or you can run the `DISABLE_DB_ACCESS` procedure in the `DBMS_AQADM` package. For example, suppose you want to disable the `oe` user from performing queue operations on the `strm01_queue` created in "Creating an ANYDATA Queue" on page 12-2.

> **Attention:** If an AQ agent is used for multiple secure queues, then running `DISABLE_DB_ACCESS` for the agent prevents the user associated with the agent from performing operations on all of these queues.

1. Run the following procedure to disable the `oe` user from performing queue operations on the secure queue `strm01_queue`:

```
BEGIN
  DBMS_AQADM.DISABLE_DB_ACCESS(
    agent_name  => 'strm01_queue_agent',
    db_username => 'oe');
END;
/
```

2. If the agent is no longer needed, you can drop the agent:

```
BEGIN
  DBMS_AQADM.DROP_AQ_AGENT(
    agent_name  => 'strm01_queue_agent');
END;
/
```

3. Revoke privileges on the queue from the user, if the user no longer needs these privileges.

```
BEGIN
  DBMS_AQADM.REVOKE_QUEUE_PRIVILEGE (
   privilege   => 'ALL',
   queue_name  => 'strmadmin.strm01_queue',
   grantee     => 'oe');
END;
```

```
/
```

See Also:

- "Secure Queues" on page 3-23
- *Oracle Database PL/SQL Packages and Types Reference* for more information about AQ agents and using the DBMS_AQADM package

Removing an ANYDATA Queue

You use the REMOVE_QUEUE procedure in the DBMS_STREAMS_ADM package to remove an existing ANYDATA queue. When you run the REMOVE_QUEUE procedure, it waits until any existing **message**s in the **queue** are consumed. Next, it stops the queue, which means that no further enqueues into the queue or dequeues from the queue are allowed. When the queue is stopped, it drops the queue.

You can also drop the **queue table** for the queue if it is empty and is not used by another queue. To do so, specify true, the default, for the drop_unused_queue_table parameter.

In addition, you can drop any **Streams client**s that use the queue by setting the cascade parameter to true. By default, the cascade parameter is set to false.

For example, to remove an ANYDATA queue named strm01_queue in the strmadmin schema and drop its empty queue table, run the following procedure:

```
BEGIN
  DBMS_STREAMS_ADM.REMOVE_QUEUE(
    queue_name              => 'strmadmin.strm01_queue',
    cascade                 => false,
    drop_unused_queue_table => true);
END;
/
```

In this case, because the cascade parameter is set to false, this procedure drops the strm01_queue only if no Streams clients use the queue. If the cascade parameter is set to false and any Streams client uses the queue, then an error is raised.

Managing Streams Propagations and Propagation Jobs

A **propagation** propagates **message**s from a Streams **source queue** to a Streams **destination queue**. This section provides instructions for completing the following tasks:

- Creating a Propagation Between Two ANYDATA Queues
- Starting a Propagation
- Stopping a Propagation
- Altering the Schedule of a Propagation Job
- Specifying the Rule Set for a Propagation
- Adding Rules to the Rule Set for a Propagation
- Removing a Rule from the Rule Set for a Propagation
- Removing a Rule Set for a Propagation
- Dropping a Propagation

In addition, you can use the features of Oracle Advanced Queuing (AQ) to manage Streams propagations.

See Also:

- "Message Propagation Between Queues" on page 3-4
- *Oracle Streams Advanced Queuing User's Guide and Reference* for more information about managing propagations with the features of AQ

Creating a Propagation Between Two ANYDATA Queues

You can use any of the following procedures to create a **propagation** between two ANYDATA queues:

- DBMS_STREAMS_ADM.ADD_SUBSET_PROPAGATION_RULES
- DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES
- DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES
- DBMS_STREAMS_ADM.ADD_GLOBAL_PROPAGATION_RULES
- DBMS_PROPAGATION_ADM.CREATE_PROPAGATION

Each of these procedures in the DBMS_STREAMS_ADM package creates a propagation with the specified name if it does not already exist, creates either a **positive rule set** or **negative rule set** for the propagation if the propagation does not have such a **rule set**, and can add **table rule**s, **schema rule**s, or **global rule**s to the rule set. The CREATE_PROPAGATION procedure creates a propagation, but does not create a rule set or **rule**s for the propagation. However, the CREATE_PROPAGATION procedure enables you to specify an existing rule set to associate with the propagation, either as a positive or a negative rule set. All propagations are started automatically upon creation.

The following tasks must be completed before you create a propagation:

- Create a **source queue** and a **destination queue** for the propagation, if they do not exist. See "Creating an ANYDATA Queue" on page 12-2 for instructions.
- Create a database link between the database containing the source queue and the database containing the destination queue. See "Configuring a Streams Administrator" on page 10-1 for information.

Example of Creating a Propagation Using DBMS_STREAMS_ADM

The following example runs the ADD_TABLE_PROPAGATION_RULES procedure in the DBMS_STREAMS_ADM package to create a propagation:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES(
    table_name              => 'hr.departments',
    streams_name            => 'strm01_propagation',
    source_queue_name       => 'strmadmin.strm_a_queue',
    destination_queue_name  => 'strmadmin.strm_b_queue@dbs2.net',
    include_dml             => true,
    include_ddl             => true,
    include_tagged_lcr      => false,
    source_database         => 'dbs1.net',
    inclusion_rule          => true,
    queue_to_queue          => true);
END;
/
```

Running this procedure performs the following actions:

- Creates a propagation named `strm01_propagation`. The propagation is created only if it does not already exist.
- Specifies that the propagation propagates LCRs from `strm_a_queue` in the current database to `strm_b_queue` in the `dbs2.net` database.
- Specifies that the propagation uses the `dbs2.net` database link to propagate the LCRs, because the `destination_queue_name` parameter contains `@dbs2.net`.
- Creates a positive rule set and associates it with the propagation because the `inclusion_rule` parameter is set to `true`. The rule set uses the **evaluation context** `SYS.STREAMS$_EVALUATION_CONTEXT`. The rule set name is system generated.
- Creates two rules. One rule evaluates to `TRUE` for row LCRs that contain the results of DML changes to the `hr.departments` table. The other rule evaluates to `TRUE` for DDL LCRs that contain DDL changes to the `hr.departments` table. The rule names are system generated.
- Adds the two rules to the positive rule set associated with the propagation. The rules are added to the positive rule setbecause the `inclusion_rule` parameter is set to `true`.
- Specifies that the propagation propagates an LCR only if it has a `NULL` **tag**, because the `include_tagged_lcr` parameter is set to `false`. This behavior is accomplished through the **system-created rule**s for the propagation.
- Specifies that the **source database** for the LCRs being propagated is `dbs1.net`, which might or might not be the current database. This propagation does not propagate LCRs in the source queue that have a different source database.
- Creates a **propagation job** for the queue-to-queue propagation.

Note: To use queue-to-queue propagation, the compatibility level must be 10.2.0 or higher for each database that contains a queue involved in the propagation.

See Also:

- "Message Propagation Between Queues" on page 3-4
- "System-Created Rules" on page 6-5
- "Queue-to-Queue Propagations" on page 3-5
- *Oracle Streams Replication Administrator's Guide* for more information about Streams tags

Example of Creating a Propagation Using DBMS_PROPAGATION_ADM

The following example runs the CREATE_PROPAGATION procedure in the DBMS_PROPAGATION_ADM package to create a propagation:

```
BEGIN
  DBMS_PROPAGATION_ADM.CREATE_PROPAGATION(
    propagation_name   => 'strm02_propagation',
    source_queue       => 'strmadmin.strm03_queue',
    destination_queue  => 'strmadmin.strm04_queue',
    destination_dblink => 'dbs2.net',
    rule_set_name      => 'strmadmin.strm01_rule_set',
    queue_to_queue     => true);
END;
/
```

Running this procedure performs the following actions:

- Creates a propagation named strm02_propagation. A propagation with the same name must not exist.
- Specifies that the propagation propagates **message**s from strm03_queue in the current database to strm04_queue in the dbs2.net database. Depending on the rules in the rule sets for the propagation, the propagated messages can be **captured message**s or **user-enqueued message**s, or both.
- Specifies that the propagation uses the dbs2.net database link to propagate the messages.
- Associates the propagation with an existing rule set named strm01_rule_set. This rule set is the positive rule set for the propagation.
- Creates a **propagation job** for the queue-to-queue propagation.

Note: To use queue-to-queue propagation, the compatibility level must be 10.2.0 or higher for each database that contains a queue involved in the propagation.

See Also:

- "Captured and User-Enqueued Messages in an ANYDATA Queue" on page 3-3
- "Message Propagation Between Queues" on page 3-4
- "Queue-to-Queue Propagations" on page 3-5

Starting a Propagation

You run the START_PROPAGATION procedure in the DBMS_PROPAGATION_ADM package to start an existing **propagation**. For example, the following procedure starts a propagation named strm01_propagation:

```
BEGIN
  DBMS_PROPAGATION_ADM.START_PROPAGATION(
    propagation_name => 'strm01_propagation');
END;
/
```

Stopping a Propagation

You run the STOP_PROPAGATION procedure in the DBMS_PROPAGATION_ADM package to stop an existing **propagation**. For example, the following procedure stops a propagation named strm01_propagation:

```
BEGIN
  DBMS_PROPAGATION_ADM.STOP_PROPAGATION(
    propagation_name => 'strm01_propagation',
    force            => false);
END;
/
```

To clear the statistics for the propagation when it is stopped, set the force parameter to true. If there is a problem with a propagation, then stopping the propagation with the force parameter set to true and restarting the propagation might correct the problem. If the force parameter is set to false, then the statistics for the propagation are not cleared.

Altering the Schedule of a Propagation Job

To alter the schedule of an existing **propagation job**, use the ALTER_PROPAGATION_SCHEDULE procedure in the DBMS_AQADM package. The following sections contain examples that alter the schedule of a propagation job for a queue-to-queue **propagation** and for a queue-to-dblink propagation. These examples set the propagation job to propagate **message**s every 15 minutes (900 seconds), with each propagation lasting 300 seconds, and a 25-second wait before new messages in a completely propagated **queue** are propagated.

See Also:

- *Oracle Streams Advanced Queuing User's Guide and Reference* for more information about using the ALTER_PROPAGATION_SCHEDULE procedure
- "Queue-to-Queue Propagations" on page 3-5
- "Propagation Jobs" on page 3-22

Altering the Schedule of a Propagation Job for a Queue-to-Queue Propagation

To alter the schedule of a propagation job for a queue-to-queue propagation that propagates messages from the strmadmin.strm_a_queue source queue to the strmadmin.strm_b_queue **destination queue** using the dbs2.net database link, run the following procedure:

```
BEGIN
  DBMS_AQADM.ALTER_PROPAGATION_SCHEDULE(
   queue_name        => 'strmadmin.strm_a_queue',
   destination       => 'dbs2.net',
   duration          => 300,
   next_time         => 'SYSDATE + 900/86400',
   latency           => 25,
   destination_queue => 'strmadmin.strm_b_queue');
END;
/
```

Because each queue-to-queue propagation has its own propagation job, this procedure alters only the schedule of the propagation that propagates messages between the two queues specified. The destination_queue parameter must specify the name of the destination queue to alter the **propagation schedule** of a queue-to-queue propagation.

Altering the Schedule of a Propagation Job for a Queue-to-Dblink Propagation

To alter the schedule of a propagation job for a queue-to-dblink propagation that propagates messages from the strmadmin.strm01_queue source queue using the dbs3.net database link, run the following procedure:

```
BEGIN
  DBMS_AQADM.ALTER_PROPAGATION_SCHEDULE(
   queue_name  => 'strmadmin.strm01_queue',
   destination => 'dbs3.net',
   duration    => 300,
   next_time   => 'SYSDATE + 900/86400',
   latency     => 25);
END;
/
```

Because the propagation is a queue-to-dblink propagation, the destination_queue parameter is not specified. Completing this task affects all queue-to-dblink propagations that propagate messages from the source queue to all destination queues that use the dbs3.net database link.

Specifying the Rule Set for a Propagation

You can specify one **positive rule set** and one **negative rule set** for a **propagation**. The propagation propagates a **message** if it evaluates to TRUE for at least one **rule** in the positive rule set and discards a change if it evaluates to TRUE for at least one rule in the negative rule set. The negative rule set is evaluated before the positive rule set.

See Also:

- Chapter 5, "Rules"
- Chapter 6, "How Rules Are Used in Streams"

Specifying a Positive Rule Set for a Propagation

You specify an existing **rule set** as the **positive rule set** for an existing **propagation** using the rule_set_name parameter in the ALTER_PROPAGATION procedure. This procedure is in the DBMS_PROPAGATION_ADM package.

For example, the following procedure sets the positive rule set for a propagation named strm01_propagation to strm02_rule_set.

```
BEGIN
  DBMS_PROPAGATION_ADM.ALTER_PROPAGATION(
    propagation_name  => 'strm01_propagation',
    rule_set_name     => 'strmadmin.strm02_rule_set');
END;
/
```

Specifying a Negative Rule Set for a Propagation

You specify an existing **rule set** as the **negative rule set** for an existing **propagation** using the `negative_rule_set_name` parameter in the `ALTER_PROPAGATION` procedure. This procedure is in the `DBMS_PROPAGATION_ADM` package.

For example, the following procedure sets the negative rule set for a propagation named `strm01_propagation` to `strm03_rule_set`.

```
BEGIN
  DBMS_PROPAGATION_ADM.ALTER_PROPAGATION(
    propagation_name        => 'strm01_propagation',
    negative_rule_set_name  => 'strmadmin.strm03_rule_set');
END;
/
```

Adding Rules to the Rule Set for a Propagation

To add **rule**s to the **rule set** of a **propagation**, you can run one of the following procedures:

- `DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES`
- `DBMS_STREAMS_ADM.ADD_SUBSET_PROPAGATION_RULES`
- `DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES`
- `DBMS_STREAMS_ADM.ADD_GLOBAL_PROPAGATION_RULES`

Excluding the `ADD_SUBSET_PROPAGATION_RULES` procedure, these procedures can add rules to the **positive rule set** or **negative rule set** for a propagation. The `ADD_SUBSET_PROPAGATION_RULES` procedure can add rules only to the positive rule set for a propagation.

See Also:

- "Message Propagation Between Queues" on page 3-4
- "System-Created Rules" on page 6-5

Adding Rules to the Positive Rule Set for a Propagation

The following example runs the `ADD_TABLE_PROPAGATION_RULES` procedure in the `DBMS_STREAMS_ADM` package to add rules to the positive rule set of an existing propagation named `strm01_propagation`:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES(
    table_name              => 'hr.locations',
    streams_name            => 'strm01_propagation',
    source_queue_name       => 'strmadmin.strm_a_queue',
    destination_queue_name  => 'strmadmin.strm_b_queue@dbs2.net',
    include_dml             => true,
    include_ddl             => true,
    source_database         => 'dbs1.net',
    inclusion_rule          => true);
END;
/
```

Running this procedure performs the following actions:

- Creates two rules. One rule evaluates to TRUE for row LCRs that contain the results of DML changes to the hr.locations table. The other rule evaluates to TRUE for DDL LCRs that contain DDL changes to the hr.locations table. The rule names are system generated.
- Specifies that both rules evaluate to TRUE only for LCRs whose changes originated at the dbs1.net **source database**.
- Adds the two rules to the positive rule set associated with the propagation because the inclusion_rule parameter is set to true.

Adding Rules to the Negative Rule Set for a Propagation

The following example runs the ADD_TABLE_PROPAGATION_RULES procedure in the DBMS_STREAMS_ADM package to add rules to the negative rule set of an existing propagation named strm01_propagation:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES(
    table_name              => 'hr.departments',
    streams_name            => 'strm01_propagation',
    source_queue_name       => 'strmadmin.strm_a_queue',
    destination_queue_name  => 'strmadmin.strm_b_queue@dbs2.net',
    include_dml             => true,
    include_ddl             => true,
    source_database         => 'dbs1.net',
    inclusion_rule          => false);
END;
/
```

Running this procedure performs the following actions:

- Creates two rules. One rule evaluates to TRUE for row LCRs that contain the results of DML changes to the hr.departments table, and the other rule evaluates to TRUE for DDL LCRs that contain DDL changes to the hr.departments table. The rule names are system generated.
- Specifies that both rules evaluate to TRUE only for LCRs whose changes originated at the dbs1.net source database.
- Adds the two rules to the negative rule set associated with the propagation because the inclusion_rule parameter is set to false.

Removing a Rule from the Rule Set for a Propagation

You remove a **rule** from the **rule set** for an existing **propagation** by running the REMOVE_RULE procedure in the DBMS_STREAMS_ADM package. For example, the following procedure removes a rule named departments3 from the **positive rule set** of a propagation named strm01_propagation.

```
BEGIN
  DBMS_STREAMS_ADM.REMOVE_RULE(
    rule_name        => 'departments3',
    streams_type     => 'propagation',
    streams_name     => 'strm01_propagation',
    drop_unused_rule => true,
    inclusion_rule   => true);
END;
/
```

In this example, the drop_unused_rule parameter in the REMOVE_RULE procedure is set to true, which is the default setting. Therefore, if the rule being removed is not in any other rule set, then it will be dropped from the database. If the drop_unused_rule parameter is set to false, then the rule is removed from the rule set, but it is not dropped from the database even if it is not in any other rule set.

If the inclusion_rule parameter is set to false, then the REMOVE_RULE procedure removes the rule from the **negative rule set** for the propagation, not the positive rule set.

To remove all of the rules in the rule set for the propagation, then specify NULL for the rule_name parameter when you run the REMOVE_RULE procedure.

> **See Also:** "Streams Client with One or More Empty Rule Sets" on page 6-4

Removing a Rule Set for a Propagation

You specify that you want to remove a **rule set** from a **propagation** using the ALTER_PROPAGATION procedure in the DBMS_PROPAGATION_ADM package. This procedure can remove the **positive rule set**, **negative rule set**, or both. Specify true for the remove_rule_set parameter to remove the positive rule set for the propagation. Specify true for the remove_negative_rule_set parameter to remove the negative rule set for the propagation.

For example, the following procedure removes both the positive and the negative rule set from a propagation named strm01_propagation.

```
BEGIN
  DBMS_PROPAGATION_ADM.ALTER_PROPAGATION(
    propagation_name         => 'strm01_propagation',
    remove_rule_set          => true,
    remove_negative_rule_set => true);
END;
/
```

> **Note:** If a propagation does not have a positive or negative rule set, then the propagation propagates all **message**s in the **source queue** to the **destination queue**.

Dropping a Propagation

You run the DROP_PROPAGATION procedure in the DBMS_PROPAGATION_ADM package to drop an existing **propagation**. For example, the following procedure drops a propagation named strm01_propagation:

```
BEGIN
  DBMS_PROPAGATION_ADM.DROP_PROPAGATION(
    propagation_name      => 'strm01_propagation',
    drop_unused_rule_sets => true);
END;
/
```

Because the drop_unused_rule_sets parameter is set to true, this procedure also drops any **rule set**s used by the propagation strm01_propagation, unless a rule set is used by another **Streams client**. If the drop_unused_rule_sets parameter is set to true, then both the **positive rule set** and **negative rule set** for the propagation

might be dropped. If this procedure drops a rule set, then it also drops any **rule**s in the rule set that are not in another rule set.

Note: When you drop a propagation, the **propagation job** used by the propagation is dropped automatically, if no other propagations are using the propagation job.

Managing a Streams Messaging Environment

Streams enables messaging with **queue**s of type ANYDATA. These queues stage **user message**s whose payloads are of ANYDATA type, and an ANYDATA payload can be a wrapper for payloads of different datatypes.

This section provides instructions for completing the following tasks:

- Wrapping User Message Payloads in an ANYDATA Wrapper and Enqueuing Them
- Dequeuing a Payload that Is Wrapped in an ANYDATA Payload
- Configuring a Messaging Client and Message Notification

Note: The examples in this section assume that you have configured a Streams administrator at each database.

See Also:

- "ANYDATA Queues and User Messages" on page 3-11 for conceptual information about messaging in Streams
- "Configuring a Streams Administrator" on page 10-1
- *Oracle Streams Advanced Queuing User's Guide and Reference* for more information about AQ
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the ANYDATA type

Wrapping User Message Payloads in an ANYDATA Wrapper and Enqueuing Them

You can wrap almost any type of payload in an ANYDATA payload. The following sections provide examples of enqueuing **message**s into, and dequeuing messages from, an ANYDATA queue.

The following steps illustrate how to wrap payloads of various types in an ANYDATA payload.

1. Connect as an administrative user who can create users, grant privileges, create tablespaces, and alter users at the dbs1.net database.
2. Grant EXECUTE privilege on the DBMS_AQ package to the oe user so that this user can run the ENQUEUE and DEQUEUE procedures in that package:

   ```
   GRANT EXECUTE ON DBMS_AQ TO oe;
   ```

3. Connect as the Streams administrator, as in the following example:

   ```
   CONNECT strmadmin/strmadminpw@dbs1.net
   ```

4. Create an ANYDATA queue if one does not already exist.

```
BEGIN
  DBMS_STREAMS_ADM.SET_UP_QUEUE(
    queue_table  => 'oe_q_table_any',
    queue_name   => 'oe_q_any',
    queue_user   => 'oe');
END;
/
```

The oe user is configured automatically as a **secure queue** user of the oe_q_any **queue** and is given ENQUEUE and DEQUEUE privileges on the queue. In addition, an AQ agent named oe is configured and is associated with the oe user. However, a message cannot be enqueued into a queue unless a subscriber who can dequeue the message is configured.

5. Add a subscriber for oe_q_any queue. This subscriber will perform explicit dequeues of messages.

```
DECLARE
  subscriber SYS.AQ$_AGENT;
BEGIN
  subscriber :=  SYS.AQ$_AGENT('OE', NULL, NULL);
  SYS.DBMS_AQADM.ADD_SUBSCRIBER(
    queue_name  =>  'strmadmin.oe_q_any',
    subscriber  =>  subscriber);
END;
/
```

6. Connect as the oe user.

```
CONNECT oe/oe@dbs1.net
```

7. Create a procedure that takes as an input parameter an object of ANYDATA type and enqueues a message containing the payload into an existing ANYDATA queue.

```
CREATE OR REPLACE PROCEDURE oe.enq_proc (payload ANYDATA)
IS
  enqopt     DBMS_AQ.ENQUEUE_OPTIONS_T;
  mprop      DBMS_AQ.MESSAGE_PROPERTIES_T;
  enq_msgid  RAW(16);
BEGIN
  mprop.SENDER_ID := SYS.AQ$_AGENT('OE', NULL, NULL);
  DBMS_AQ.ENQUEUE(
    queue_name          =>  'strmadmin.oe_q_any',
    enqueue_options     =>  enqopt,
    message_properties  =>  mprop,
    payload             =>  payload,
    msgid               =>  enq_msgid);
END;
/
```

8. Run the procedure you created in Step 7 by specifying the appropriate Convert*data_type* function. The following commands enqueue messages of various types.

VARCHAR2 type:

```
EXEC oe.enq_proc(ANYDATA.ConvertVarchar2('Chemicals - SW'));
COMMIT;
```

NUMBER type:

```
EXEC oe.enq_proc(ANYDATA.ConvertNumber('16'));
COMMIT;
```

User-defined type:

```
BEGIN
  oe.enq_proc(ANYDATA.ConvertObject(oe.cust_address_typ(
    '1646 Brazil Blvd','361168','Chennai','Tam', 'IN')));
END;
/
COMMIT;
```

> **See Also:** "Viewing the Contents of User-Enqueued Messages in a Queue" on page 21-4 for information about viewing the contents of these enqueued messages

Dequeuing a Payload that Is Wrapped in an ANYDATA Payload

The following steps illustrate how to dequeue a payload wrapped in an ANYDATA payload. This example assumes that you have completed the steps in "Wrapping User Message Payloads in an ANYDATA Wrapper and Enqueuing Them" on page 12-15.

To dequeue **message**s, you must know the consumer of the messages. To find the consumer for the messages in a **queue**, connect as the owner of the queue and query the AQ$*queue_table_name*, where *queue_table_name* is the name of the queue table. For example, to find the consumers of the messages in the oe_q_any queue, run the following query:

```
CONNECT strmadmin/strmadminpw@dbs1.net

SELECT MSG_ID, MSG_STATE, CONSUMER_NAME FROM AQ$OE_Q_TABLE_ANY;
```

1. Connect as the oe user:

   ```
   CONNECT oe/oe@dbs1.net
   ```

2. Create a procedure that takes as an input the consumer of the messages you want to dequeue. The following example procedure dequeues messages of oe.cust_address_typ and prints the contents of the messages.

   ```
   CREATE OR REPLACE PROCEDURE oe.get_cust_address (
   consumer IN VARCHAR2) AS
     address         OE.CUST_ADDRESS_TYP;
     deq_address     ANYDATA;
     msgid           RAW(16);
     deqopt          DBMS_AQ.DEQUEUE_OPTIONS_T;
     mprop           DBMS_AQ.MESSAGE_PROPERTIES_T;
     new_addresses   BOOLEAN := true;
     next_trans      EXCEPTION;
     no_messages     EXCEPTION;
     pragma exception_init (next_trans, -25235);
     pragma exception_init (no_messages, -25228);
     num_var         pls_integer;
   ```

```
BEGIN
     deqopt.consumer_name := consumer;
     deqopt.wait := 1;
     WHILE (new_addresses) LOOP
     BEGIN
      DBMS_AQ.DEQUEUE(
         queue_name          =>  'strmadmin.oe_q_any',
         dequeue_options     =>  deqopt,
         message_properties  =>  mprop,
         payload             =>  deq_address,
         msgid               =>  msgid);
      deqopt.navigation := DBMS_AQ.NEXT;
      DBMS_OUTPUT.PUT_LINE('****');
      IF (deq_address.GetTypeName() = 'OE.CUST_ADDRESS_TYP') THEN
          DBMS_OUTPUT.PUT_LINE('Message TYPE is: ' ||
                               deq_address.GetTypeName());
          num_var := deq_address.GetObject(address);
          DBMS_OUTPUT.PUT_LINE(' **** CUSTOMER ADDRESS **** ');
          DBMS_OUTPUT.PUT_LINE(address.street_address);
          DBMS_OUTPUT.PUT_LINE(address.postal_code);
          DBMS_OUTPUT.PUT_LINE(address.city);
          DBMS_OUTPUT.PUT_LINE(address.state_province);
          DBMS_OUTPUT.PUT_LINE(address.country_id);
      ELSE
         DBMS_OUTPUT.PUT_LINE('Message TYPE is: ' ||
                              deq_address.GetTypeName());
      END IF;
    COMMIT;
    EXCEPTION
      WHEN next_trans THEN
      deqopt.navigation := DBMS_AQ.NEXT_TRANSACTION;
      WHEN no_messages THEN
        new_addresses := false;
        DBMS_OUTPUT.PUT_LINE('No more messages');
     END;
  END LOOP;
END;
/
```

3. Run the procedure you created in Step 1 and specify the consumer of the messages you want to dequeue, as in the following example:

```
SET SERVEROUTPUT ON SIZE 100000
EXEC oe.get_cust_address('OE');
```

Configuring a Messaging Client and Message Notification

This section contains instructions for configuring the following elements in a database:

- An enqueue procedure that enqueues **message**s into an `ANYDATA` queue at a database. In this example, the enqueue procedure uses a trigger to enqueue a message every time a row is inserted into the `oe.orders` table.
- A **messaging client** that can dequeue **user-enqueued message**s based on **rule**s. In this example, the messaging client uses a rule so that it dequeues only messages that involve the `oe.orders` table. The messaging client uses the `DEQUEUE` procedure in the `DBMS_STREAMS_MESSAGING` to dequeue one message at a time and display the order number for the order.

- Message notification for the messaging client. In this example, a notification is sent to an email address when a message is enqueued into the **queue** used by the messaging client. The message can be dequeued by the messaging client because the message satisfies the **rule set**s of the messaging client.

You can query the DBA_STREAMS_MESSAGE_CONSUMERS data dictionary view for information about existing messaging clients and notifications.

Complete the following steps to configure a messaging client and message notification:

1. Connect as an administrative user who can grant privileges and execute subprograms in supplied packages.
2. Set the host name used to send the email, the mail port, and the email account that sends email messages for email notifications using the DBMS_AQELM package. The following example sets the mail host name to smtp.mycompany.com, the mail port to 25, and the email account to Mary.Smith@mycompany.com:

```
BEGIN
  DBMS_AQELM.SET_MAILHOST('smtp.mycompany.com') ;
  DBMS_AQELM.SET_MAILPORT(25) ;
  DBMS_AQELM.SET_SENDFROM('Mary.Smith@mycompany.com');
END;
/
```

   You can use procedures in the DBMS_AQELM package to determine the current mail host, mail port, and send from settings for a database. For example, to determine the current mail host for a database, use the DBMS_AQELM.GET_MAILHOST procedure.

3. Grant the necessary privileges to the users who will create the messaging client, enqueue and dequeue messages, and specify message notifications. In this example, the oe user performs all of these tasks.

```
GRANT EXECUTE ON DBMS_AQ TO oe;
GRANT EXECUTE ON DBMS_STREAMS_ADM TO oe;
GRANT EXECUTE ON DBMS_STREAMS_MESSAGING TO oe;

BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_RULE_SET_OBJ,
    grantee      => 'oe',
    grant_option => false);
END;
/

BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_RULE_OBJ,
    grantee      => 'oe',
    grant_option => false);
END;
/

BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT_OBJ,
    grantee      => 'oe',
    grant_option => false);
END;
/
```

4. Connect as the oe user:

```
CONNECT oe/oe
```

5. Create an ANYDATA queue using SET_UP_QUEUE, as in the following example:

```
BEGIN
  DBMS_STREAMS_ADM.SET_UP_QUEUE(
    queue_table  => 'oe.notification_queue_table',
    queue_name   => 'oe.notification_queue');
END;
/
```

6. Create the types for the user-enqueued messages, as in the following example:

```
CREATE TYPE oe.user_msg AS OBJECT(
  object_name    VARCHAR2(30),
  object_owner   VARCHAR2(30),
  message        VARCHAR2(50));
/
```

7. Create a trigger that enqueues a message into the queue whenever an order is inserted into the oe.orders table, as in the following example:

```
CREATE OR REPLACE TRIGGER oe.order_insert AFTER INSERT
ON oe.orders FOR EACH ROW
DECLARE
  msg            oe.user_msg;
  str            VARCHAR2(2000);
BEGIN
  str := 'New Order - ' || :NEW.ORDER_ID || ' Order ID';
  msg  := oe.user_msg(
             object_name   => 'ORDERS',
             object_owner  => 'OE',
             message       => str);
  DBMS_STREAMS_MESSAGING.ENQUEUE (
    queue_name   => 'oe.notification_queue',
    payload      => ANYDATA.CONVERTOBJECT(msg));
END;
/
```

8. Create the messaging client that will dequeue messages from the queue and the rule used by the messaging client to determine which messages to dequeue, as in the following example:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_MESSAGE_RULE (
    message_type   => 'oe.user_msg',
    rule_condition => ' :msg.OBJECT_OWNER = ''OE'' AND  ' ||
                      ' :msg.OBJECT_NAME = ''ORDERS'' ',
    streams_type   => 'dequeue',
    streams_name   => 'oe',
    queue_name     => 'oe.notification_queue');
END;
/
```

9. Set the message notification to send email upon enqueue of messages that can be dequeued by the messaging client, as in the following example:

```
BEGIN
  DBMS_STREAMS_ADM.SET_MESSAGE_NOTIFICATION (
    streams_name         => 'oe',
    notification_action  => 'Mary.Smith@mycompany.com',
    notification_type    => 'MAIL',
    include_notification => true,
    queue_name           => 'oe.notification_queue');
END;
/
```

10. Create a PL/SQL procedure that dequeues messages using the messaging client, as in the following example:

```
CREATE OR REPLACE PROCEDURE oe.deq_notification(consumer IN VARCHAR2) AS
  msg            ANYDATA;
  user_msg       oe.user_msg;
  num_var        PLS_INTEGER;
  more_messages  BOOLEAN := true;
  navigation     VARCHAR2(30);
BEGIN
  navigation := 'FIRST MESSAGE';
  WHILE (more_messages) LOOP
    BEGIN
      DBMS_STREAMS_MESSAGING.DEQUEUE(
        queue_name   => 'oe.notification_queue',
        streams_name => consumer,
        payload      => msg,
        navigation   => navigation,
        wait         => DBMS_STREAMS_MESSAGING.NO_WAIT);
      IF msg.GETTYPENAME() = 'OE.USER_MSG' THEN
        num_var := msg.GETOBJECT(user_msg);
        DBMS_OUTPUT.PUT_LINE(user_msg.object_name);
        DBMS_OUTPUT.PUT_LINE(user_msg.object_owner);
        DBMS_OUTPUT.PUT_LINE(user_msg.message);
      END IF;
      navigation := 'NEXT MESSAGE';
      COMMIT;
    EXCEPTION WHEN SYS.DBMS_STREAMS_MESSAGING.ENDOFCURTRANS THEN
                navigation := 'NEXT TRANSACTION';
              WHEN DBMS_STREAMS_MESSAGING.NOMOREMSGS THEN
                more_messages := false;
                DBMS_OUTPUT.PUT_LINE('No more messages.');
              WHEN OTHERS THEN
                RAISE;
    END;
  END LOOP;
END;
/
```

11. Insert rows into the `oe.orders` table, as in the following example:

```
INSERT INTO oe.orders VALUES(2521, 'direct', 144, 0, 922.57, 159, NULL);
INSERT INTO oe.orders VALUES(2522, 'direct', 116, 0, 1608.29, 153, NULL);
COMMIT;
INSERT INTO oe.orders VALUES(2523, 'direct', 116, 0, 227.55, 155, NULL);
COMMIT;
```

Message notification sends a message to the email address specified in Step 9 for each message that was enqueued. Each notification is an `AQXmlNotification`, which includes of the following:

- `notification_options`, which includes the following:
  - `destination` - The **destination queue** from which the message was dequeued
  - `consumer_name` - The name of the messaging client that dequeued the message
- `message_set` - The set of message properties

The following example shows the `AQXmlNotification` format sent in an email notification:

```
<?xml version="1.0" encoding="UTF-8"?>
<Envelope xmlns="http://ns.oracle.com/AQ/schemas/envelope">
    <Body>
        <AQXmlNotification xmlns="http://ns.oracle.com/AQ/schemas/access">
            <notification_options>
                <destination>OE.NOTIFICATION_QUEUE</destination>
                <consumer_name>OE</consumer_name>
            </notification_options>
            <message_set>
                <message>
                    <message_header>
                        <message_id>CB510DDB19454731E034080020AE3E0A</message_id>
                        <expiration>-1</expiration>
                        <delay>0</delay>
                        <priority>1</priority>
                        <delivery_count>0</delivery_count>
                        <sender_id>
                            <agent_name>OE</agent_name>
                            <protocol>0</protocol>
                        </sender_id>
                        <message_state>0</message_state>
                    </message_header>
                </message>
            </message_set>
        </AQXmlNotification>
    </Body>
</Envelope>
```

You can dequeue the messages enqueued in this example by running the `oe.deq_notification` procedure:

```
SET SERVEROUTPUT ON SIZE 100000
EXEC oe.deq_notification('OE');
```

See Also:

- "Viewing the Messaging Clients in a Database" on page 21-2
- "Viewing Message Notifications" on page 21-3
- Chapter 6, "How Rules Are Used in Streams" for more information about rule sets for **Streams client**s and for information about how messages satisfy rule sets
- *Oracle Streams Advanced Queuing User's Guide and Reference* and *Oracle XML DB Developer's Guide* for more information about message notifications and XML

13

Managing an Apply Process

A Streams **apply process** dequeues logical change records (LCRs) and **user message**s from a specific **queue** and either applies each one directly or passes it as a parameter to a user-defined procedure.

This chapter contains these topics:

Each task described in this chapter should be completed by a Streams administrator that has been granted the appropriate privileges, unless specified otherwise.

See Also:

- Chapter 4, "Streams Apply Process"
- "Configuring a Streams Administrator" on page 10-1
- *Oracle Streams Replication Administrator's Guide* for more information about managing **DML handler**s, **DDL handler**s, and Streams tags for an apply process

Creating an Apply Process

You can use any of the following procedures to create an **apply process**:

- DBMS_STREAMS_ADM.ADD_TABLE_RULES
- DBMS_STREAMS_ADM.ADD_SUBSET_RULES
- DBMS_STREAMS_ADM.ADD_SCHEMA_RULES
- DBMS_STREAMS_ADM.ADD_GLOBAL_RULES
- DBMS_STREAMS_ADM.ADD_MESSAGE_RULE
- DBMS_APPLY_ADM.CREATE_APPLY

Each of the procedures in the DBMS_STREAMS_ADM package creates an apply process with the specified name if it does not already exist, creates either a **positive rule set** or **negative rule set** for the apply process if the apply process does not have such a **rule set**, and can add **table rule**s, **schema rule**s, **global rule**s, or a **message rule** to the rule set.

The CREATE_APPLY procedure in the DBMS_APPLY_ADM package creates an apply process, but does not create a rule set or **rule**s for the apply process. However, the CREATE_APPLY procedure enables you to specify an existing rule set to associate with the apply process, either as a positive or a negative rule set, and a number of other options, such as **apply handler**s, an **apply user**, an apply **tag**, and whether to apply **captured message**s or **user-enqueued message**s.

Before you create an apply process, create an ANYDATA queue to associate with the apply process, if one does not exist.

Note:

- Depending on the configuration of the apply process you create, **supplemental logging** might be required at the **source database** on columns in the tables for which an apply process applies changes.
- To create an apply process, a user must be granted DBA role.

See Also:

- "Creating an ANYDATA Queue" on page 12-2
- "Supplemental Logging in a Streams Environment" on page 2-11 for information about supplemental logging
- "Specifying Supplemental Logging at a Source Database" on page 11-29

Examples of Creating an Apply Process Using DBMS_STREAMS_ADM

The first example in this section creates an **apply process** that applies **captured message**s. The second example in this section creates an apply process that applies **user-enqueued message**s. A single apply process cannot apply both captured and user-enqueued messages.

- Creating an Apply Process for Captured Messages
- Creating an Apply Process for User-Enqueued Messages

See Also:

- "Apply Process Creation" on page 4-13
- "System-Created Rules" on page 6-5
- *Oracle Streams Replication Administrator's Guide* for more information about Streams tags

Creating an Apply Process for Captured Messages

The following example runs the `ADD_SCHEMA_RULES` procedure in the `DBMS_STREAMS_ADM` package to create an apply process that applies captured messages:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
    schema_name         => 'hr',
    streams_type        => 'apply',
    streams_name        => 'strm01_apply',
    queue_name          => 'strm01_queue',
    include_dml         => true,
    include_ddl         => false,
    include_tagged_lcr  => false,
    source_database     => 'dbs1.net',
    inclusion_rule      => true);
END;
/
```

Running this procedure performs the following actions:

- Creates an apply process named `strm01_apply` that applies captured messages to the local database. The apply process is created only if it does not already exist.
- Associates the apply process with an existing **queue** named `strm01_queue`.
- Creates a **positive rule set** and associates it with the apply process, if the apply process does not have a positive rule set, because the `inclusion_rule` parameter is set to `true`. The rule set uses the `SYS.STREAMS$_EVALUATION_CONTEXT` **evaluation context**. The rule set name is system generated.
- Creates one **rule** that evaluates to `TRUE` for row LCRs that contain the results of DML changes to database objects in the `hr` schema. The rule name is system generated.
- Adds the rule to the positive rule set associated with the apply process because the `inclusion_rule` parameter is set to `true`.
- Sets the `apply_tag` for the apply process to a value that is the hexadecimal equivalent of `'00'` (double zero). Redo entries generated by the apply process have a **tag** with this value.
- Specifies that the apply process applies a row LCR only if it has a `NULL` tag, because the `include_tagged_lcr` parameter is set to `false`. This behavior is accomplished through the **system-created rule** for the apply process.
- Specifies that the LCRs applied by the apply process originate at the `dbs1.net` **source database**. The rules in the apply process rule sets determine which messages are dequeued by the apply process. If the apply process dequeues an LCR with a source database other than `dbs1.net`, then an error is raised.

Creating an Apply Process for User-Enqueued Messages

The following example runs the ADD_MESSAGE_RULE procedure in the DBMS_STREAMS_ADM package to create an apply process:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_MESSAGE_RULE(
    message_type       => 'oe.order_typ',
    rule_condition     => ':msg.order_status = 1',
    streams_type       => 'apply',
    streams_name       => 'strm02_apply',
    queue_name         => 'strm02_queue',
    inclusion_rule     => true);
END;
/
```

Running this procedure performs the following actions:

- Creates an apply process named strm02_apply that dequeues user-enqueued messages of oe.order_typ type and sends them to the **message handler** for the apply process. The apply process is created only if it does not already exist.
- Associates the apply process with an existing queue named strm02_queue.
- Creates a positive rule set and associates it with the apply process, if the apply process does not have a positive rule set, because the inclusion_rule parameter is set to true. The rule set name is system generated, and the rule set does not use an **evaluation context**.
- Creates one rule that evaluates to TRUE for user-enqueued messages that satisfy the **rule condition**. The rule uses a system-created evaluation context for the message type. The rule name and the evaluation context name are system generated.
- Adds the rule to the positive rule set associated with the apply process because the inclusion_rule parameter is set to true.
- Sets the apply_tag for the apply process to a value that is the hexadecimal equivalent of '00' (double zero). Redo entries generated by the apply process, including any redo entries generated by a message handler, have a tag with this value.

Note: You can use the ALTER_APPLY procedure in the DBMS_APPLY_ADM package to specify a message handler for an apply process.

See Also:

- "Message Rule Example" on page 6-27
- "Evaluation Contexts for Message Rules" on page 6-35

Examples of Creating an Apply Process Using DBMS_APPLY_ADM

The first example in this section creates an **apply process** that applies **captured message**s. The second example in this section creates an apply process that applies **user-enqueued message**s. A single apply process cannot apply both captured and user-enqueued messages.

- Creating an Apply Process for Captured Messages with DBMS_APPLY_ADM
- Creating an Apply Process for User-Enqueued Messages with DBMS_APPLY_ADM

See Also:

- "Apply Process Creation" on page 4-13
- "Message Processing Options for an Apply Process" on page 4-3 for more information about apply handlers
- *Oracle Streams Replication Administrator's Guide* for more information about Streams tags
- *Oracle Streams Replication Administrator's Guide* for information about configuring an apply process to apply messages to a non-Oracle database using the `apply_database_link` parameter

Creating an Apply Process for Captured Messages with DBMS_APPLY_ADM

The following example runs the `CREATE_APPLY` procedure in the `DBMS_APPLY_ADM` package to create an apply process that applies captured messages:

```
BEGIN
  DBMS_APPLY_ADM.CREATE_APPLY(
    queue_name             => 'strm03_queue',
    apply_name             => 'strm03_apply',
    rule_set_name          => 'strmadmin.strm03_rule_set',
    message_handler        => NULL,
    ddl_handler            => 'strmadmin.history_ddl',
    apply_user             => 'hr',
    apply_database_link    => NULL,
    apply_tag              => HEXTORAW('5'),
    apply_captured         => true,
    precommit_handler      => NULL,
    negative_rule_set_name => NULL,
    source_database        => 'dbs1.net');
END;
/
```

Running this procedure performs the following actions:

- Creates an apply process named `strm03_apply`. An apply process with the same name must not exist.
- Associates the apply process with an existing **queue** named `strm03_queue`.
- Associates the apply process with an existing **rule set** named `strm03_rule_set`. This rule set is the **positive rule set** for the apply process.
- Specifies that the apply process does not use a **message handler**.

- Specifies that the **DDL handler** is the `history_ddl` PL/SQL procedure in the `strmadmin` schema. The user who runs the `CREATE_APPLY` procedure must have `EXECUTE` privilege on the `history_ddl` PL/SQL procedure. An example in the *Oracle Streams Replication Administrator's Guide* creates this procedure.
- Specifies that the user who applies the changes is `hr`, and not the user who is running the `CREATE_APPLY` procedure (the Streams administrator).
- Specifies that the apply process applies changes to the local database because the `apply_database_link` parameter is set to `NULL`.
- Specifies that each redo entry generated by the apply process has a **tag** that is the hexadecimal equivalent of `'5'`.
- Specifies that the apply process applies captured messages, and not user-enqueued messages. Therefore, if an LCR that was constructed by a user application, not by a **capture process**, is staged in the queue for the apply process, then this apply process does not apply the LCR.
- Specifies that the apply process does not use a **precommit handler**.
- Specifies that the apply process does not use a **negative rule set**.
- Specifies that the LCRs applied by the apply process originate at the `dbs1.net` **source database**. The **rule**s in the apply process rule sets determine which messages are dequeued by the apply process. If the apply process dequeues an LCR with a source database other than `dbs1.net`, then an error is raised.

Creating an Apply Process for User-Enqueued Messages with DBMS_APPLY_ADM

The following example runs the `CREATE_APPLY` procedure in the `DBMS_APPLY_ADM` package to create an apply process that applies user-enqueued messages:

```
BEGIN
  DBMS_APPLY_ADM.CREATE_APPLY(
    queue_name             => 'strm04_queue',
    apply_name             => 'strm04_apply',
    rule_set_name          => 'strmadmin.strm04_rule_set',
    message_handler        => 'strmadmin.mes_handler',
    ddl_handler            => NULL,
    apply_user             => NULL,
    apply_database_link    => NULL,
    apply_tag              => NULL,
    apply_captured         => false,
    precommit_handler      => NULL,
    negative_rule_set_name => NULL);
END;
/
```

Running this procedure performs the following actions:

- Creates an apply process named `strm04_apply`. An apply process with the same name must not exist.
- Associates the apply process with an existing queue named `strm04_queue`.
- Associates the apply process with an existing rule set named `strm04_rule_set`. This rule set is the positive rule set for the apply process.
- Specifies that the message handler is the `mes_handler` PL/SQL procedure in the `strmadmin` schema. The user who runs the `CREATE_APPLY` procedure must have `EXECUTE` privilege on the `mes_handler` PL/SQL procedure.

- Specifies that the apply process does not use a DDL handler.
- Specifies that the user who applies the changes is the user who runs the CREATE_APPLY procedure, because the apply_user parameter is NULL.
- Specifies that the apply process applies changes to the local database, because the apply_database_link parameter is set to NULL.
- Specifies that each redo entry generated by the apply process has a NULL tag.
- Specifies that the apply process applies user-enqueued messages, and not captured messages.
- Specifies that the apply process does not use a **precommit handler**.
- Specifies that the apply process does not use a negative rule set.

Starting an Apply Process

You run the START_APPLY procedure in the DBMS_APPLY_ADM package to start an existing **apply process**. For example, the following procedure starts an apply process named strm01_apply:

```
BEGIN
  DBMS_APPLY_ADM.START_APPLY(
    apply_name => 'strm01_apply');
END;
/
```

Stopping an Apply Process

You run the STOP_APPLY procedure in the DBMS_APPLY_ADM package to stop an existing **apply process**. For example, the following procedure stops an apply process named strm01_apply:

```
BEGIN
  DBMS_APPLY_ADM.STOP_APPLY(
    apply_name => 'strm01_apply');
END;
/
```

Managing the Rule Set for an Apply Process

This section contains instructions for completing the following tasks:

- Specifying the Rule Set for an Apply Process
- Adding Rules to the Rule Set for an Apply Process
- Removing a Rule from the Rule Set for an Apply Process
- Removing a Rule Set for an Apply Process

See Also:

- Chapter 5, "Rules"
- Chapter 6, "How Rules Are Used in Streams"

Specifying the Rule Set for an Apply Process

You can specify one **positive rule set** and one **negative rule set** for an **apply process**. The apply process applies a **message** if it evaluates to TRUE for at least one **rule** in the positive rule set and discards a message if it evaluates to TRUE for at least one rule in the negative rule set. The negative rule set is evaluated before the positive rule set.

Specifying a Positive Rule Set for an Apply Process

You specify an existing **rule set** as the positive rule set for an existing apply process using the rule_set_name parameter in the ALTER_APPLY procedure. This procedure is in the DBMS_APPLY_ADM package.

For example, the following procedure sets the positive rule set for an apply process named strm01_apply to strm02_rule_set.

```
BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name    => 'strm01_apply',
    rule_set_name => 'strmadmin.strm02_rule_set');
END;
/
```

Specifying a Negative Rule Set for an Apply Process

You specify an existing rule set as the negative rule set for an existing apply process using the negative_rule_set_name parameter in the ALTER_APPLY procedure. This procedure is in the DBMS_APPLY_ADM package.

For example, the following procedure sets the negative rule set for an apply process named strm01_apply to strm03_rule_set.

```
BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name             => 'strm01_apply',
    negative_rule_set_name => 'strmadmin.strm03_rule_set');
END;
/
```

Adding Rules to the Rule Set for an Apply Process

To add **rule**s to the **rule set** for an **apply process**, you can run one of the following procedures:

- DBMS_STREAMS_ADM.ADD_TABLE_RULES
- DBMS_STREAMS_ADM.ADD_SUBSET_RULES
- DBMS_STREAMS_ADM.ADD_SCHEMA_RULES
- DBMS_STREAMS_ADM.ADD_GLOBAL_RULES

Excluding the ADD_SUBSET_RULES procedure, these procedures can add rules to the **positive rule set** or **negative rule set** for an apply process. The ADD_SUBSET_RULES procedure can add rules only to the positive rule set for an apply process.

See Also: "System-Created Rules" on page 6-5

Adding Rules to the Positive Rule Set for an Apply Process

The following example runs the ADD_TABLE_RULES procedure in the DBMS_STREAMS_ADM package to add rules to the positive rule set of an apply process named strm01_apply:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name       => 'hr.departments',
    streams_type     => 'apply',
    streams_name     => 'strm01_apply',
    queue_name       => 'strm01_queue',
    include_dml      => true,
    include_ddl      => true,
    source_database  => 'dbs1.net',
    inclusion_rule   => true);
END;
/
```

Running this procedure performs the following actions:

- Creates one rule that evaluates to `TRUE` for row LCRs that contain the results of DML changes to the `hr.departments` table. The rule name is system generated.
- Creates one rule that evaluates to `TRUE` for DDL LCRs that contain DDL changes to the `hr.departments` table. The rule name is system generated.
- Specifies that both rules evaluate to `TRUE` only for LCRs whose changes originated at the `dbs1.net` **source database**.
- Adds the rules to the positive rule set associated with the apply process because the `inclusion_rule` parameter is set to `true`.

Adding Rules to the Negative Rule Set for an Apply Process

The following example runs the `ADD_TABLE_RULES` procedure in the `DBMS_STREAMS_ADM` package to add rules to the negative rule set of an apply process named `strm01_apply`:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name       => 'hr.regions',
    streams_type     => 'apply',
    streams_name     => 'strm01_apply',
    queue_name       => 'strm01_queue',
    include_dml      => true,
    include_ddl      => true,
    source_database  => 'dbs1.net',
    inclusion_rule   => false);
END;
/
```

Running this procedure performs the following actions:

- Creates one rule that evaluates to `TRUE` for row LCRs that contain the results of DML changes to the `hr.regions` table. The rule name is system generated.
- Creates one rule that evaluates to `TRUE` for DDL LCRs that contain DDL changes to the `hr.regions` table. The rule name is system generated.
- Specifies that both rules evaluate to `TRUE` only for LCRs whose changes originated at the `dbs1.net` source database.
- Adds the rules to the negative rule set associated with the apply process because the `inclusion_rule` parameter is set to `false`.

Removing a Rule from the Rule Set for an Apply Process

You remove a **rule** from a **rule set** for an existing **apply process** by running the REMOVE_RULE procedure in the DBMS_STREAMS_ADM package. For example, the following procedure removes a rule named departments3 from the **positive rule set** of an apply process named strm01_apply.

```
BEGIN
  DBMS_STREAMS_ADM.REMOVE_RULE(
    rule_name        => 'departments3',
    streams_type     => 'apply',
    streams_name     => 'strm01_apply',
    drop_unused_rule => true,
    inclusion_rule   => true);
END;
/
```

In this example, the drop_unused_rule parameter in the REMOVE_RULE procedure is set to true, which is the default setting. Therefore, if the rule being removed is not in any other rule set, then it will be dropped from the database. If the drop_unused_rule parameter is set to false, then the rule is removed from the rule set, but it is not dropped from the database even if it is not in any other rule set.

If the inclusion_rule parameter is set to false, then the REMOVE_RULE procedure removes the rule from the **negative rule set** for the apply process, not from the positive rule set.

To remove all of the rules in a rule set for the apply process, then specify NULL for the rule_name parameter when you run the REMOVE_RULE procedure.

> **See Also:** "Streams Client with One or More Empty Rule Sets" on page 6-4

Removing a Rule Set for an Apply Process

You remove a **rule set** from an existing **apply process** using the ALTER_APPLY procedure in the DBMS_APPLY_ADM package. This procedure can remove the **positive rule set**, **negative rule set**, or both. Specify true for the remove_rule_set parameter to remove the positive rule set for the apply process. Specify true for the remove_negative_rule_set parameter to remove the negative rule set for the apply process.

For example, the following procedure removes both the positive and negative rule sets from an apply process named strm01_apply.

```
BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name              => 'strm01_apply',
    remove_rule_set         => true,
    remove_negative_rule_set => true);
END;
/
```

Note: If an apply process that applies **captured message**s does not have a positive or negative rule set, then the apply process applies all captured messages in its **queue**. Similarly, if an apply process that applies user-enqueued messages does not have a positive or negative rule set, then the apply process applies all user-enqueued messages in its queue.

Setting an Apply Process Parameter

Set an **apply process** parameter using the `SET_PARAMETER` procedure in the `DBMS_APPLY_ADM` package. Apply process parameters control the way an apply process operates.

For example, the following procedure sets the `commit_serialization` parameter for an apply process named `strm01_apply` to `none`. This setting for the `commit_serialization` parameter enables the apply process to commit transactions in any order.

```
BEGIN
  DBMS_APPLY_ADM.SET_PARAMETER(
    apply_name   => 'strm01_apply',
    parameter    => 'commit_serialization',
    value        => 'none');
END;
/
```

Note:

- The `value` parameter is always entered as a `VARCHAR2` value, even if the parameter value is a number.
- If you set the `parallelism` apply process parameter to a value greater than `1`, then you must specify a conditional **supplemental log group** at the **source database** for all of the unique key and foreign key columns in the tables for which an apply process applies changes. **supplemental logging** might be required for other columns in these tables as well, depending on your configuration.

See Also:

- "Apply Process Parameters" on page 4-14
- The `DBMS_APPLY_ADM.SET_PARAMETER` procedure in the *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the apply process parameters
- "Specifying Supplemental Logging at a Source Database" on page 11-29

Setting the Apply User for an Apply Process

The **apply user** is the user who applies all DML changes and DDL changes that satisfy the **apply process rule set**s and who runs user-defined **apply handler**s. Set the apply user for an apply process using the `apply_user` parameter in the `ALTER_APPLY` procedure in the `DBMS_APPLY_ADM` package.

To change the apply user, the user who invokes the `ALTER_APPLY` procedure must be granted `DBA` role. Only the `SYS` user can set the `apply_user` to `SYS`.

For example, the following procedure sets the apply user for an apply process named `strm03_apply` to `hr`.

```
BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name => 'strm03_apply',
    apply_user => 'hr');
END;
/
```

Running this procedure grants the new apply user dequeue privilege on the **queue** used by the apply process and configures the user as a **secure queue** user of the queue. In addition, make sure the apply user has the following privileges:

- `EXECUTE` privilege on the rule sets used by the apply process
- `EXECUTE` privilege on all **custom rule-based transformation** functions used in the rule set
- `EXECUTE` privilege on all apply handler procedures

These privileges must be granted directly to the apply user. They cannot be granted through roles.

Managing the Message Handler for an Apply Process

The following sections contain instructions for setting and removing the **message handler** for an **apply process**:

- Setting the Message Handler for an Apply Process
- Removing the Message Handler for an Apply Process

See Also:

- "Message Processing with an Apply Process" on page 4-2
- *Oracle Streams Advanced Queuing User's Guide and Reference* for an example that creates a **message handler**

Setting the Message Handler for an Apply Process

Set the **message handler** for an **apply process** using the `message_handler` parameter in the `ALTER_APPLY` procedure in the `DBMS_APPLY_ADM` package. For example, the following procedure sets the message handler for an apply process named `strm03_apply` to the `mes_handler` procedure in the `oe` schema.

```
BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name      => 'strm03_apply',
    message_handler => 'oe.mes_handler');
END;
/
```

The user who runs the `ALTER_APPLY` procedure must have `EXECUTE` privilege on the specified message handler.

Removing the Message Handler for an Apply Process

You remove the **message handler** for an **apply process** by setting the `remove_message_handler` parameter to `true` in the `ALTER_APPLY` procedure in the `DBMS_APPLY_ADM` package. For example, the following procedure removes the message handler from an apply process named `strm03_apply`.

```
BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name             => 'strm03_apply',
    remove_message_handler => true);
END;
/
```

Managing the Precommit Handler for an Apply Process

The following sections contain instructions for creating, setting, and removing the **precommit handler** for an **apply process**:

- Creating a Precommit Handler for an Apply Process
- Setting the Precommit Handler for an Apply Process
- Removing the Precommit Handler for an Apply Process

Creating a Precommit Handler for an Apply Process

A **precommit handler** must have the following signature:

```
PROCEDURE handler_procedure (
   parameter_name   IN  NUMBER);
```

Here, *`handler_procedure`* stands for the name of the procedure and *`parameter_name`* stands for the name of the parameter passed to the procedure. The parameter passed to the procedure is a commit SCN from an internal commit directive in the **queue** used by the **apply process**.

You can use a precommit handler to record information about commits processed by an apply process. The apply process can apply **captured message**s or **user-enqueued message**s. For a captured row LCR, a commit directive contains the commit SCN of the transaction from the **source database**. For a user-enqueued message, the commit SCN is generated by the apply process.

The precommit handler procedure must conform to the following restrictions:

- Any work that commits must be an autonomous transaction.
- Any rollback must be to a named savepoint created in the procedure.

If a precommit handler raises an exception, then the entire apply transaction is rolled back, and all of the messages in the transaction are moved to the error queue.

For example, a precommit handler can be used for auditing the row LCRs applied by an apply process. Such a precommit handler is used with one or more separate **DML handler**s to record the source database commit SCN for a transaction, and possibly the time when the apply process applies the transaction, in an audit table.

Specifically, this example creates a precommit handler that is used with a DML handler that records information about row LCRs in the following table:

```
CREATE TABLE strmadmin.history_row_lcrs(
  timestamp             DATE,
  source_database_name  VARCHAR2(128),
  command_type          VARCHAR2(30),
  object_owner          VARCHAR2(32),
  object_name           VARCHAR2(32),
  tag                   RAW(10),
  transaction_id        VARCHAR2(10),
  scn                   NUMBER,
  commit_scn            NUMBER,
  old_values            SYS.LCR$_ROW_LIST,
  new_values            SYS.LCR$_ROW_LIST)
    NESTED TABLE old_values STORE AS old_values_ntab
    NESTED TABLE new_values STORE AS new_values_ntab;
```

The DML handler inserts a row in the `strmadmin.history_row_lcrs` table for each row LCR processed by an apply process. The precommit handler created in this example inserts a row into the `strmadmin.history_row_lcrs` table when a transaction commits.

Create the procedure that inserts the commit information into the `history_row_lcrs` table:

```
CREATE OR REPLACE PROCEDURE strmadmin.history_commit(commit_number IN NUMBER)
 IS
 BEGIN
  -- Insert commit information into the history_row_lcrs table
  INSERT INTO strmadmin.history_row_lcrs (timestamp, commit_scn)
    VALUES (SYSDATE, commit_number);
END;
/
```

See Also:

- "Audit Commit Information for Messages Using Precommit Handlers" on page 4-6
- *Oracle Streams Replication Administrator's Guide* for more information about the DML handler referenced in this example

Setting the Precommit Handler for an Apply Process

A **precommit handler** processes all commit directives dequeued by an **apply process**.

Set the precommit handler for an apply process using the `precommit_handler` parameter in the `ALTER_APPLY` procedure in the `DBMS_APPLY_ADM` package. For example, the following procedure sets the precommit handler for an apply process named `strm01_apply` to the `history_commit` procedure in the `strmadmin` schema.

```
BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name        => 'strm01_apply',
    precommit_handler => 'strmadmin.history_commit');
END;
/
```

You can also specify a precommit handler when you create an apply process using the `CREATE_APPLY` procedure in the `DBMS_APPLY_ADM` package.

Removing the Precommit Handler for an Apply Process

You remove the **precommit handler** for an **apply process** by setting the `remove_precommit_handler` parameter to `true` in the `ALTER_APPLY` procedure in the `DBMS_APPLY_ADM` package. For example, the following procedure removes the precommit handler from an apply process named `strm01_apply`.

```
BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name               => 'strm01_apply',
    remove_precommit_handler => true);
END;
/
```

Specifying Message Enqueues by Apply Processes

This section contains instructions for setting a destination **queue** into which **apply process**es that use a specified **rule** in a **positive rule set** will enqueue **message**s that satisfy the rule. This section also contains instructions for removing destination queue settings.

> **See Also:** "Viewing Rules that Specify a Destination Queue on Apply" on page 22-14

Setting the Destination Queue for Messages that Satisfy a Rule

You use the `SET_ENQUEUE_DESTINATION` procedure in the `DBMS_APPLY_ADM` package to set a destination **queue** for **message**s that satisfy a specific **rule**. For example, to set the destination queue for a rule named `employees5` to the queue `hr.change_queue`, run the following procedure:

```
BEGIN
  DBMS_APPLY_ADM.SET_ENQUEUE_DESTINATION(
    rule_name               =>  'employees5',
    destination_queue_name  =>  'hr.change_queue');
END;
/
```

This procedure modifies the **action context** of the rule to specify the queue. Any **apply process** in the local database with the `employees5` rule in its **positive rule set** will enqueue a message into `hr.change_queue` if the message satisfies the `employees5` rule. If you want to change the destination queue for the `employees5` rule, then run the `SET_ENQUEUE_DESTINATION` procedure again and specify a different queue.

The **apply user** of each apply process using the specified rule must have the necessary privileges to enqueue messages into the specified queue. If the queue is a **secure queue**, then the apply user must be a secure queue user of the queue.

A message that has been enqueued into an queue using the `SET_ENQUEUE_DESTINATION` procedure is the same as any other **user-enqueued message**. Such messages can be manually dequeued, applied by an apply process created with the `apply_captured` parameter set to `false`, or propagated to another queue.

Note: The specified rule must be in the positive rule set for an apply process. If the rule is in the **negative rule set** for an apply process, then the apply process does not enqueue the message into the destination queue.

See Also:

- "Enabling a User to Perform Operations on a Secure Queue" on page 12-3
- "Enqueue Destinations for Messages During Apply" on page 6-39 for more information about how the SET_ENQUEUE_DESTINATION procedure modifies the action context of the specified rule

Removing the Destination Queue Setting for a Rule

You use the SET_ENQUEUE_DESTINATION procedure in the DBMS_APPLY_ADM package to remove a destination **queue** for **message**s that satisfy a specified **rule**. Specifically, you set the destination_queue_name parameter in this procedure to NULL for the rule. When a destination queue specification is removed for a rule, messages that satisfy the rule are no longer enqueued into the queue by an **apply process**.

For example, to remove the destination queue for a rule named employees5, run the following procedure:

```
BEGIN
  DBMS_APPLY_ADM.SET_ENQUEUE_DESTINATION(
    rule_name               =>  'employees5',
    destination_queue_name  =>  NULL);
END;
/
```

Any apply process in the local database with the employees5 rule in its **positive rule set** no longer enqueues a message into hr.change_queue if the message satisfies the employees5 rule.

Specifying Execute Directives for Apply Processes

This section contains instructions for setting an **apply process** execute directive for **message**s that satisfy a specified **rule** in the **positive rule set** for the apply process.

See Also: "Viewing Rules that Specify No Execution on Apply" on page 22-15

Specifying that Messages that Satisfy a Rule Are Not Executed

You use the SET_EXECUTE procedure in the DBMS_APPLY_ADM package to specify that **apply process**es do not execute **message**s that satisfy a specified **rule**. Specifically, you set the execute parameter in this procedure to false for the rule. After setting the execution directive to false for a rule, an apply process with the rule in its **positive rule set** does not execute a message that satisfies the rule.

For example, to specify that apply processes do not execute messages that satisfy a rule named departments8, run the following procedure:

```
BEGIN
  DBMS_APPLY_ADM.SET_EXECUTE(
    rule_name   =>  'departments8',
    execute     =>  false);
END;
/
```

This procedure modifies the **action context** of the rule to specify the execution directive. Any apply process in the local database with the `departments8` rule in its positive rule set will not execute a message if the message satisfies the `departments8` rule. That is, if the message is an LCR, then an apply process does not apply the change in the LCR to the relevant database object. Also, an apply process does not send a message that satisfies this rule to any **apply handler**.

Note:

- The specified rule must be in the positive rule set for an apply process for the apply process to follow the execution directive. If the rule is in the **negative rule set** for an apply process, then the apply process ignores the execution directive for the rule.
- The `SET_EXECUTE` procedure can be used with the `SET_ENQUEUE_DESTINATION` procedure if you want to enqueue messages that satisfy a particular rule into a destination **queue** without executing these messages. After a message is enqueued using the `SET_ENQUEUE_DESTINATION` procedure, it is a **user-enqueued message** in the destination queue. Therefore, it can be manually dequeued, applied by an apply process, or propagated to another queue.

See Also:

- "Execution Directives for Messages During Apply" on page 6-39 for more information about how the `SET_EXECUTE` procedure modifies the action context of the specified rule
- "Specifying Message Enqueues by Apply Processes" on page 13-15

Specifying that Messages that Satisfy a Rule Are Executed

You use the `SET_EXECUTE` procedure in the `DBMS_APPLY_ADM` package to specify that **apply process**es execute **message**s that satisfy a specified **rule**. Specifically, you set the `execute` parameter in this procedure to `true` for the rule. By default, each apply process executes messages that satisfy a rule in the **positive rule set** for the apply process, assuming that the message does not satisfy a rule in the **negative rule set** for the apply process. Therefore, you need to set the `execute` parameter to `true` for a rule only if this parameter was set to `false` for the rule in the past.

For example, to specify that apply processes executes messages that satisfy a rule named `departments8`, run the following procedure:

```
BEGIN
  DBMS_APPLY_ADM.SET_EXECUTE(
    rule_name   =>  'departments8',
    execute     =>  true);
END;
/
```

Any apply process in the local database with the `departments8` rule in its positive rule set will execute a message if the message satisfies the `departments8` rule. That is, if the message is an LCR, then an apply process applies the change in the LCR to the relevant database object. Also, an apply process sends a message that satisfies this rule to an **apply handler** if it is configured to do so.

Managing an Error Handler

The following sections contain instructions for creating, setting, and removing an **error handler**:

- Creating an Error Handler
- Setting an Error Handler
- Unsetting an Error Handler

See Also: "Message Processing with an Apply Process" on page 4-2

Creating an Error Handler

You create an **error handler** by running the `SET_DML_HANDLER` procedure in the `DBMS_APPLY_ADM` package and setting the `error_handler` parameter to `true`.

An error handler must have the following signature:

```
PROCEDURE user_procedure (
     message             IN ANYDATA,
     error_stack_depth   IN NUMBER,
     error_numbers       IN DBMS_UTILITY.NUMBER_ARRAY,
     error_messages      IN emsg_array);
```

Here, *`user_procedure`* stands for the name of the procedure. Each parameter is required and must have the specified datatype. However, you can change the names of the parameters. The `emsg_array` parameter must be a user-defined array that is a PL/SQL table of type `VARCHAR2` with at least 76 characters.

Note: Some conditions on the user procedure specified in `SET_DML_HANDLER` must be met for error handlers. See *Oracle Streams Replication Administrator's Guide* for information about these conditions.

Running an error handler results in one of the following outcomes:

- The error handler successfully resolves the error, applies the row LCR if appropriate, and returns control back to the **apply process**.
- The error handler fails to resolve the error, and the error is raised. The raised error causes the transaction to be rolled back and placed in the error queue.

If you want to retry the DML operation, then have the error handler procedure run the `EXECUTE` member procedure for the LCR.

The following example creates an error handler named `regions_pk_error` that resolves primary key violations for the `hr.regions` table. At a **destination database**, assume users insert rows into the `hr.regions` table and an apply process applies changes to the `hr.regions` table that originated from a **capture process** at a remote **source database**. In this environment, there is a possibility of errors resulting from users at the destination database inserting a row with the same primary key value as an insert row LCR applied from the source database.

This example creates a table in the strmadmin schema called errorlog to record the following information about each primary key violation error on the hr.regions table:

- The timestamp when the error occurred
- The name of the apply process that raised the error
- The user who caused the error (sender), which is the capture process name for **captured message**s or the name of the AQ agent for user-enqueued LCRs
- The name of the object on which the DML operation was run, because errors for other objects might be logged in the future
- The type of command used in the DML operation
- The name of the constraint violated
- The error message
- The LCR that caused the error

This error handler resolves only errors that are caused by a primary key violation on the hr.regions table. To resolve this type of error, the error handler modifies the region_id value in the row LCR using a sequence and then executes the row LCR to apply it. If other types of errors occur, then you can use the row LCR you stored in the errorlog table to resolve the error manually.

For example, the following error is resolved by the error handler:

1. At the destination database, a user inserts a row into the hr.regions table with a region_id value of 6 and a region_name value of 'LILLIPUT'.
2. At the source database, a user inserts a row into the hr.regions table with a region_id value of 6 and a region_name value of 'BROBDINGNAG'.
3. A capture process at the source database captures the change described in Step 2.
4. A **propagation** propagates the LCR containing the change from a queue at the source database to the queue used by the apply process at the destination database.
5. When the apply process tries to apply the LCR, an error results because of a primary key violation.
6. The apply process invokes the error handler to handle the error.
7. The error handler logs the error in the strmadmin.errorlog table.
8. The error handler modifies the region_id value in the LCR using a sequence and executes the LCR to apply it.

Complete the following steps to create the regions_pk_error error handler:

1. Create the sequence used by the error handler to assign new primary key values by connecting as hr user and running the following statement:

   ```
   CONNECT hr/hr

   CREATE SEQUENCE hr.reg_exception_s START WITH 9000;
   ```

   This example assumes that users at the destination database will never insert a row into the hr.regions table with a region_id greater than 8999.

2. Grant the Streams administrator ALL privilege on the sequence:

   ```
   GRANT ALL ON reg_exception_s TO strmadmin;
   ```

3. Create the `errorlog` table by connecting as the Streams administrator and running the following statement:

```
CONNECT strmadmin/strmadminpw

CREATE TABLE strmadmin.errorlog(
  logdate       DATE,
  apply_name    VARCHAR2(30),
  sender        VARCHAR2(100),
  object_name   VARCHAR2(32),
  command_type  VARCHAR2(30),
  errnum        NUMBER,
  errmsg        VARCHAR2(2000),
  text          VARCHAR2(2000),
  lcr           SYS.LCR$_ROW_RECORD);
```

4. Create a package that includes the `regions_pk_error` procedure:

```
CREATE OR REPLACE PACKAGE errors_pkg
AS
 TYPE emsg_array IS TABLE OF VARCHAR2(2000) INDEX BY BINARY_INTEGER;
 PROCEDURE regions_pk_error(
   message            IN ANYDATA,
   error_stack_depth  IN NUMBER,
   error_numbers      IN DBMS_UTILITY.NUMBER_ARRAY,
   error_messages     IN EMSG_ARRAY);
END errors_pkg ;
/
```

5. Create the package body:

```
CREATE OR REPLACE PACKAGE BODY errors_pkg AS
 PROCEDURE regions_pk_error (
   message            IN ANYDATA,
   error_stack_depth  IN NUMBER,
   error_numbers      IN DBMS_UTILITY.NUMBER_ARRAY,
   error_messages     IN EMSG_ARRAY )
 IS
  reg_id     NUMBER;
  ad         ANYDATA;
  lcr        SYS.LCR$_ROW_RECORD;
  ret        PLS_INTEGER;
  vc         VARCHAR2(30);
  apply_name VARCHAR2(30);
  errlog_rec errorlog%ROWTYPE ;
  ov2        SYS.LCR$_ROW_LIST;
 BEGIN
  -- Access the error number from the top of the stack.
  -- In case of check constraint violation,
  -- get the name of the constraint violated.
  IF error_numbers(1) IN ( 1 , 2290 ) THEN
   ad  := DBMS_STREAMS.GET_INFORMATION('CONSTRAINT_NAME');
   ret := ad.GetVarchar2(errlog_rec.text);
  ELSE
   errlog_rec.text := NULL ;
  END IF ;
  -- Get the name of the sender and the name of the apply process.
  ad  := DBMS_STREAMS.GET_INFORMATION('SENDER');
  ret := ad.GETVARCHAR2(errlog_rec.sender);
  apply_name := DBMS_STREAMS.GET_STREAMS_NAME();
```

```
  -- Try to access the LCR.
  ret := message.GETOBJECT(lcr);
  errlog_rec.object_name  := lcr.GET_OBJECT_NAME() ;
  errlog_rec.command_type := lcr.GET_COMMAND_TYPE() ;
  errlog_rec.errnum := error_numbers(1) ;
  errlog_rec.errmsg := error_messages(1) ;
  INSERT INTO strmadmin.errorlog VALUES (SYSDATE, apply_name,
       errlog_rec.sender, errlog_rec.object_name, errlog_rec.command_type,
       errlog_rec.errnum, errlog_rec.errmsg, errlog_rec.text, lcr);
  -- Add the logic to change the contents of LCR with correct values.
  -- In this example, get a new region_id number
  -- from the hr.reg_exception_s sequence.
  ov2 := lcr.GET_VALUES('new', 'n');
  FOR i IN 1 .. ov2.count
  LOOP
    IF ov2(i).column_name = 'REGION_ID' THEN
     SELECT hr.reg_exception_s.NEXTVAL INTO reg_id FROM DUAL;
     ov2(i).data := ANYDATA.ConvertNumber(reg_id) ;
    END IF ;
  END LOOP ;
  -- Set the NEW values in the LCR.
  lcr.SET_VALUES(value_type => 'NEW', value_list => ov2);
  -- Execute the modified LCR to apply it.
  lcr.EXECUTE(true);
 END regions_pk_error;
END errors_pkg;
/
```

Note:

- For subsequent changes to the modified row to be applied successfully, you should converge the rows at the two databases as quickly as possible. That is, you should make the `region_id` for the row match at the source and destination database. If you do not want these manual changes to be recaptured at a database, then use the `SET_TAG` procedure in the `DBMS_STREAMS` package to set the **tag** for the session in which you make the change to a value that is not captured.
- This example error handler illustrates the use of the `GET_VALUES` member function and `SET_VALUES` member procedure for the LCR. If you are modifying only one value in the LCR, then the `GET_VALUE` member function and `SET_VALUE` member procedure might be more convenient and more efficient.

See Also: *Oracle Streams Replication Administrator's Guide* for more information about setting tag values generated by the current session

Setting an Error Handler

An **error handler** handles errors resulting from a row LCR dequeued by any **apply process** that contains a specific operation on a specific table. You can specify multiple error handlers on the same table, to handle errors resulting from different operations on the table. You can either set an error handler for a specific apply process, or you can set an error handler as a general error handler that is used by all apply processes that apply the specified operation to the specified table.

Set an error handler using the `SET_DML_HANDLER` procedure in the `DBMS_APPLY_ADM` package. When you run this procedure to set an error handler, set the `error_handler` parameter to `true`.

For example, the following procedure sets the error handler for `INSERT` operations on the `hr.regions` table. Therefore, when any apply process dequeues a row LCR containing an `INSERT` operation on the local `hr.regions` table, and the row LCR results in an error, the apply process sends the row LCR to the `strmadmin.errors_pkg.regions_pk_error` PL/SQL procedure for processing. If the error handler cannot resolve the error, then the row LCR and all of the other row LCRs in the same transaction are moved to the error queue.

In this example, the `apply_name` parameter is set to `NULL`. Therefore, the error handler is a general error handler that is used by all of the apply processes in the database.

Run the following procedure to set the error handler:

```
BEGIN
  DBMS_APPLY_ADM.SET_DML_HANDLER(
    object_name         => 'hr.regions',
    object_type         => 'TABLE',
    operation_name      => 'INSERT',
    error_handler       => true,
    user_procedure      => 'strmadmin.errors_pkg.regions_pk_error',
    apply_database_link => NULL,
    apply_name          => NULL);
END;
/
```

Unsetting an Error Handler

You unset an **error handler** using the `SET_DML_HANDLER` procedure in the `DBMS_APPLY_ADM` package. When you run that procedure, set the `user_procedure` parameter to `NULL` for a specific operation on a specific table.

For example, the following procedure unsets the error handler for `INSERT` operations on the `hr.regions` table:

```
BEGIN
  DBMS_APPLY_ADM.SET_DML_HANDLER(
    object_name    => 'hr.regions',
    object_type    => 'TABLE',
    operation_name => 'INSERT',
    user_procedure => NULL,
    apply_name     => NULL);
END;
/
```

Note: The `error_handler` parameter does not need to be specified.

Managing Apply Errors

The following sections contain instructions for retrying and deleting apply errors:

- Retrying Apply Error Transactions
- Deleting Apply Error Transactions

See Also:

- "The Error Queue" on page 4-16
- "Checking for Apply Errors" on page 22-15
- "Displaying Detailed Information About Apply Errors" on page 22-16
- *Oracle Streams Replication Administrator's Guide* for information about the possible causes of apply errors

Retrying Apply Error Transactions

You can retry a specific error transaction or you can retry all error transactions for an **apply process**. You might need to make DML or DDL changes to database objects to correct the conditions that caused one or more apply errors before you retry error transactions. You can also have one or more **capture process**es configured to capture changes to the same database objects, but you might not want the changes captured. In this case, you can set the session **tag** to a value that will not be captured for the session that makes the changes.

See Also: *Oracle Streams Replication Administrator's Guide* for more information about setting tag values generated by the current session

Retrying a Specific Apply Error Transaction

When you retry an error transaction, you can execute it immediately or send the error transaction to a user procedure for modifications before executing it. The following sections provide instructions for each method:

- Retrying a Specific Apply Error Transaction Without a User Procedure
- Retrying a Specific Apply Error Transaction with a User Procedure

See Also: *Oracle Database PL/SQL Packages and Types Reference* for more information about the `EXECUTE_ERROR` procedure

Retrying a Specific Apply Error Transaction Without a User Procedure After you correct the conditions that caused an apply error, you can retry the transaction by running the `EXECUTE_ERROR` procedure in the `DBMS_APPLY_ADM` package without specifying a user procedure. In this case, the transaction is executed without any custom processing.

For example, to retry a transaction with the transaction identifier `5.4.312`, run the following procedure:

```
BEGIN
  DBMS_APPLY_ADM.EXECUTE_ERROR(
    local_transaction_id => '5.4.312',
    execute_as_user      => false,
    user_procedure       => NULL);
END;
/
```

If `execute_as_user` is `true`, then the apply process executes the transaction in the security context of the current user. If `execute_as_user` is `false`, then the apply process executes the transaction in the security context of the original receiver of the transaction. The original receiver is the user who was processing the transaction when the error was raised.

In either case, the user who executes the transaction must have privileges to perform DML and DDL changes on the apply objects and to run any **apply handler**s. This user must also have dequeue privileges on the queue used by the apply process.

Retrying a Specific Apply Error Transaction with a User Procedure You can retry an error transaction by running the `EXECUTE_ERROR` procedure in the `DBMS_APPLY_ADM` package, and specify a user procedure to modify one or more **message**s in the transaction before the transaction is executed. The modifications should enable successful execution of the transaction. The messages in the transaction can be LCRs or **user message**s.

For example, consider a case in which an apply error resulted because of a **conflict**. Examination of the error transaction reveals that the old value for the `salary` column in a row LCR contained the wrong value. Specifically, the current value of the salary of the employee with `employee_id` of `197` in the `hr.employees` table did not match the old value of the salary for this employee in the row LCR. Assume that the current value for this employee is `3250` in the `hr.employees` table.

Given this scenario, the following user procedure modifies the salary in the row LCR that caused the error:

```
CREATE OR REPLACE PROCEDURE strmadmin.modify_emp_salary(
  in_any                          IN      ANYDATA,
  error_record                    IN      DBA_APPLY_ERROR%ROWTYPE,
  error_message_number            IN      NUMBER,
  messaging_default_processing    IN OUT  BOOLEAN,
  out_any                         OUT     ANYDATA)
AS
  row_lcr          SYS.LCR$_ROW_RECORD;
  row_lcr_changed  BOOLEAN := FALSE;
  res              NUMBER;
  ob_owner         VARCHAR2(32);
  ob_name          VARCHAR2(32);
  cmd_type         VARCHAR2(30);
  employee_id      NUMBER;
BEGIN
  IF in_any.getTypeName() = 'SYS.LCR$_ROW_RECORD' THEN
    -- Access the LCR
    res := in_any.GETOBJECT(row_lcr);
    -- Determine the owner of the database object for the LCR
    ob_owner := row_lcr.GET_OBJECT_OWNER;
    -- Determine the name of the database object for the LCR
    ob_name := row_lcr.GET_OBJECT_NAME;
```

```
    -- Determine the type of DML change
    cmd_type := row_lcr.GET_COMMAND_TYPE;
    IF (ob_owner = 'HR' AND ob_name = 'EMPLOYEES' AND cmd_type = 'UPDATE') THEN
      -- Determine the employee_id of the row change
      IF row_lcr.GET_VALUE('old', 'employee_id') IS NOT NULL THEN
        employee_id := row_lcr.GET_VALUE('old', 'employee_id').ACCESSNUMBER();
        IF (employee_id = 197) THEN
          -- error_record.message_number should equal error_message_number
          row_lcr.SET_VALUE(
          value_type => 'OLD',
          column_name => 'salary',
          column_value => ANYDATA.ConvertNumber(3250));
          row_lcr_changed := TRUE;
        END IF;
      END IF;
    END IF;
  END IF;
  -- Specify that the apply process continues to process the current message
  messaging_default_processing := TRUE;
  -- assign out_any appropriately
  IF row_lcr_changed THEN
    out_any := ANYDATA.ConvertObject(row_lcr);
  ELSE
    out_any := in_any;
  END IF;
END;
/
```

To retry a transaction with the transaction identifier `5.6.924` and process the transaction with the `modify_emp_salary` procedure in the `strmadmin` schema before execution, run the following procedure:

```
BEGIN
  DBMS_APPLY_ADM.EXECUTE_ERROR(
    local_transaction_id => '5.6.924',
    execute_as_user      => false,
    user_procedure       => 'strmadmin.modify_emp_salary');
END;
/
```

Note: The user who runs the procedure must have `SELECT` privilege on `DBA_APPLY_ERROR` data dictionary view.

See Also: "Displaying Detailed Information About Apply Errors" on page 22-16

Retrying All Error Transactions for an Apply Process

After you correct the conditions that caused all of the apply errors for an apply process, you can retry all of the error transactions by running the `EXECUTE_ALL_ERRORS` procedure in the `DBMS_APPLY_ADM` package. For example, to retry all of the error transactions for an apply process named `strm01_apply`, you can run the following procedure:

```
BEGIN
  DBMS_APPLY_ADM.EXECUTE_ALL_ERRORS(
    apply_name       => 'strm01_apply',
    execute_as_user  => false);
END;
/
```

Note: If you specify `NULL` for the `apply_name` parameter, and you have multiple apply processes, then all of the apply errors are retried for all of the apply processes.

Deleting Apply Error Transactions

You can delete a specific error transaction or you can delete all error transactions for an **apply process**.

Deleting a Specific Apply Error Transaction

If an error transaction should not be applied, then you can delete the transaction from the error queue using the `DELETE_ERROR` procedure in the `DBMS_APPLY_ADM` package. For example, to delete a transaction with the transaction identifier `5.4.312`, run the following procedure:

```
EXEC DBMS_APPLY_ADM.DELETE_ERROR(local_transaction_id => '5.4.312');
```

Deleting All Error Transactions for an Apply Process

If none of the error transactions should be applied, then you can delete all of the error transactions by running the `DELETE_ALL_ERRORS` procedure in the `DBMS_APPLY_ADM` package. For example, to delete all of the error transactions for an apply process named `strm01_apply`, you can run the following procedure:

```
EXEC DBMS_APPLY_ADM.DELETE_ALL_ERRORS(apply_name => 'strm01_apply');
```

Note: If you specify `NULL` for the `apply_name` parameter, and you have multiple apply processes, then all of the apply errors are deleted for all of the apply processes.

Dropping an Apply Process

You run the `DROP_APPLY` procedure in the `DBMS_APPLY_ADM` package to drop an existing **apply process**. For example, the following procedure drops an apply process named `strm02_apply`:

```
BEGIN
  DBMS_APPLY_ADM.DROP_APPLY(
    apply_name            => 'strm02_apply',
    drop_unused_rule_sets => true);
END;
/
```

Because the `drop_unused_rule_sets` parameter is set to `true`, this procedure also drops any **rule set**s used by the `strm02_apply` apply process, unless a rule set is used by another **Streams client**. If the `drop_unused_rule_sets` parameter is set to `true`, then both the positive and **negative rule set** for the apply process might be dropped. If this procedure drops a rule set, then it also drops any **rule**s in the rule set that are not in another rule set.

An error is raised if you try to drop an apply process and there are errors in the error queue for the specified apply process. Therefore, if there are errors in the error queue for an apply process, delete the errors before dropping the apply process.

See Also: "Managing Apply Errors" on page 13-23

14

Managing Rules

A Streams environment uses **rule**s to control the behavior of **Streams client**s (**capture process**es, **propagation**s, **apply process**es, and **messaging client**s). In addition, you can create custom applications that are clients of the **rules engine**. This chapter contains instructions for managing **rule set**s, rules, and privileges related to rules.

This chapter contains these topics:

- Managing Rule Sets
- Managing Rules
- Managing Privileges on Evaluation Contexts, Rule Sets, and Rules

Each task described in this chapter should be completed by a Streams administrator that has been granted the appropriate privileges, unless specified otherwise.

Attention: Modifying the rules and rule sets used by a **Streams client** changes the behavior of the Streams client.

Note: This chapter does not contain examples for creating **evaluation context**s, nor does it contain examples for evaluating events using the `DBMS_RULE.EVALUATE` procedure. See Chapter 28, "Rule-Based Application Example" for these examples.

See Also:

- Chapter 5, "Rules"
- Chapter 6, "How Rules Are Used in Streams"
- Chapter 7, "Rule-Based Transformations"
- "Configuring a Streams Administrator" on page 10-1

Managing Rule Sets

You can modify a **rule set** without stopping Streams **capture process**es, **propagation**s, and **apply process**es that use the rule set. Streams will detect the change immediately after it is committed. If you need precise control over which **message**s use the new version of a rule set, then complete the following steps:

1. Stop the relevant capture processes, propagations, and apply processes.
2. Modify the rule set.
3. Restart the **Streams client**s you stopped in Step 1.

This section provides instructions for completing the following tasks:

- Creating a Rule Set
- Adding a Rule to a Rule Set
- Removing a Rule from a Rule Set
- Dropping a Rule Set

See Also:

- "Stopping a Capture Process" on page 11-24
- "Stopping a Propagation" on page 12-10
- "Stopping an Apply Process" on page 13-7

Creating a Rule Set

The following example runs the `CREATE_RULE_SET` procedure in the `DBMS_RULE_ADM` package to create a **rule set**:

```
BEGIN
  DBMS_RULE_ADM.CREATE_RULE_SET(
    rule_set_name       => 'strmadmin.hr_capture_rules',
    evaluation_context  => 'SYS.STREAMS$_EVALUATION_CONTEXT');
END;
/
```

Running this procedure performs the following actions:

- Creates a rule set named `hr_capture_rules` in the `strmadmin` schema. A rule set with the same name and owner must not exist.
- Associates the rule set with the `SYS.STREAMS$_EVALUATION_CONTEXT` **evaluation context**, which is the Oracle-supplied evaluation context for Streams.

You can also use the following procedures in the `DBMS_STREAMS_ADM` package to create a rule set automatically, if one does not exist for a Streams **capture process**, **propagation**, **apply process**, or **messaging client**:

- `ADD_MESSAGE_PROPAGATION_RULE`
- `ADD_MESSAGE_RULE`
- `ADD_TABLE_PROPAGATION_RULES`
- `ADD_TABLE_RULES`
- `ADD_SUBSET_PROPAGATION_RULES`
- `ADD_SUBSET_RULES`

- ADD_SCHEMA_PROPAGATION_RULES
- ADD_SCHEMA_RULES
- ADD_GLOBAL_PROPAGATION_RULES
- ADD_GLOBAL_RULES

Except for ADD_SUBSET_PROPAGATION_RULES and ADD_SUBSET_RULES, these procedures can create either a **positive rule set** or a **negative rule set** for a **Streams client**. ADD_SUBSET_PROPAGATION_RULES and ADD_SUBSET_RULES can only create a positive rule set for a Streams client.

See Also:

- "Example of Creating a Local Capture Process Using DBMS_STREAMS_ADM" on page 11-4
- "Example of Creating a Propagation Using DBMS_STREAMS_ADM" on page 12-7
- "Creating an Apply Process for Captured Messages" on page 13-3

Adding a Rule to a Rule Set

The following example runs the ADD_RULE procedure in the DBMS_RULE_ADM package to add the hr_dml **rule** to the hr_capture_rules **rule set**:

```
BEGIN
  DBMS_RULE_ADM.ADD_RULE(
    rule_name          => 'strmadmin.hr_dml',
    rule_set_name      => 'strmadmin.hr_capture_rules',
    evaluation_context => NULL);
END;
/
```

In this example, no **evaluation context** is specified when running the ADD_RULE procedure. Therefore, if the rule does not have its own evaluation context, it will inherit the evaluation context of the hr_capture_rules rule set. If you want a rule to use an evaluation context other than the one specified for the rule set, then you can set the evaluation_context parameter to this evaluation context when you run the ADD_RULE procedure.

Removing a Rule from a Rule Set

When you remove a **rule** from a **rule set**, the behavior of the **Streams client**s that use the rule set changes. Make sure you understand how removing a rule from a rule set will affect **Streams client**s before proceeding.

The following example runs the REMOVE_RULE procedure in the DBMS_RULE_ADM package to remove the hr_dml rule from the hr_capture_rules rule set:

```
BEGIN
  DBMS_RULE_ADM.REMOVE_RULE(
    rule_name     => 'strmadmin.hr_dml',
    rule_set_name => 'strmadmin.hr_capture_rules');
END;
/
```

After running the REMOVE_RULE procedure, the rule still exists in the database and, if it was in any other rule sets, it remains in those rule sets.

See Also: "Dropping a Rule" on page 14-11

Dropping a Rule Set

The following example runs the `DROP_RULE_SET` procedure in the `DBMS_RULE_ADM` package to drop the `hr_capture_rules` **rule set** from the database:

```
BEGIN
  DBMS_RULE_ADM.DROP_RULE_SET(
    rule_set_name => 'strmadmin.hr_capture_rules',
    delete_rules  => false);
END;
/
```

In this example, the `delete_rules` parameter in the `DROP_RULE_SET` procedure is set to `false`, which is the default setting. Therefore, if the rule set contains any **rule**s, then these rules are not dropped. If the `delete_rules` parameter is set to `true`, then any rules in the rule set that are not in another rule set are dropped from the database automatically. Rules in the rule set that are in one or more other rule sets are not dropped.

Managing Rules

You can modify a **rule** without stopping Streams **capture process**es, **propagation**s, and **apply process**es that use the rule. Streams will detect the change immediately after it is committed. If you need precise control over which **message**s use the new version of a rule, then complete the following steps:

1. Stop the relevant capture processes, propagations, and apply processes.
2. Modify the rule.
3. Restart the **Streams client**s you stopped in Step 1.

This section provides instructions for completing the following tasks:

- Creating a Rule
- Altering a Rule
- Modifying System-Created Rules
- Dropping a Rule

See Also:

- "Stopping a Capture Process" on page 11-24
- "Stopping a Propagation" on page 12-10
- "Stopping an Apply Process" on page 13-7

Creating a Rule

The following examples use the `CREATE_RULE` procedure in the `DBMS_RULE_ADM` package to create a **rule** without an **action context** and a rule with an action context.

Creating a Rule Without an Action Context

To create a rule without an **action context**, run the CREATE_RULE procedure and specify the rule name using the rule_name parameter and the **rule condition** using the condition parameter, as in the following example:

```
BEGIN
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name  => 'strmadmin.hr_dml',
    condition  => ' :dml.get_object_owner() = ''HR'' ');
END;
/
```

Running this procedure performs the following actions:

- Creates a rule named hr_dml in the strmadmin schema. A rule with the same name and owner must not exist.
- Creates a condition that evaluates to TRUE for any DML change to a table in the hr schema.

In this example, no **evaluation context** is specified for the rule. Therefore, the rule will either inherit the evaluation context of any **rule set** to which it is added, or it will be assigned an evaluation context explicitly when the DBMS_RULE_ADM.ADD_RULE procedure is run to add it to a rule set. At this point, the rule cannot be evaluated because it is not part of any rule set.

You can also use the following procedures in the DBMS_STREAMS_ADM package to create rules and add them to a rule set automatically:

- ADD_MESSAGE_PROPAGATION_RULE
- ADD_MESSAGE_RULE
- ADD_TABLE_PROPAGATION_RULES
- ADD_TABLE_RULES
- ADD_SUBSET_PROPAGATION_RULES
- ADD_SUBSET_RULES
- ADD_SCHEMA_PROPAGATION_RULES
- ADD_SCHEMA_RULES
- ADD_GLOBAL_PROPAGATION_RULES
- ADD_GLOBAL_RULES

Except for ADD_SUBSET_PROPAGATION_RULES and ADD_SUBSET_RULES, these procedures can add rules to either the **positive rule set** or the **negative rule set** for a **Streams client**. ADD_SUBSET_PROPAGATION_RULES and ADD_SUBSET_RULES can add rules only to the positive rule set for a Streams client.

See Also:

- "Example of Creating a Local Capture Process Using DBMS_STREAMS_ADM" on page 11-4
- "Example of Creating a Propagation Using DBMS_STREAMS_ADM" on page 12-7
- "Creating an Apply Process for Captured Messages" on page 13-3

Creating a Rule with an Action Context

To create a rule with an **action context**, run the CREATE_RULE procedure and specify the rule name using the rule_name parameter, the rule condition using the condition parameter, and the rule action context using the action_context parameter. You add a name-value pair to an action context using the ADD_PAIR member procedure of the RE$NV_LIST type

The following example creates a rule with a non-NULL action context:

```
DECLARE
  ac  SYS.RE$NV_LIST;
BEGIN
  ac := SYS.RE$NV_LIST(NULL);
  ac.ADD_PAIR('course_number', ANYDATA.CONVERTNUMBER(1057));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name      => 'strmadmin.rule_dep_10',
    condition      => ' :dml.get_object_owner()=''HR'' AND ' ||
       ' :dml.get_object_name()=''EMPLOYEES'' AND ' ||
       ' (:dml.get_value(''NEW'', ''DEPARTMENT_ID'').AccessNumber()=10) AND ' ||
       ' :dml.get_command_type() = ''INSERT'' ',
    action_context => ac);
END;
/
```

Running this procedure performs the following actions:

- Creates a rule named rule_dep_10 in the strmadmin schema. A rule with the same name and owner must not exist.
- Creates a condition that evaluates to TRUE for any insert into the hr.employees table where the department_id is 10.
- Creates an action context with one name-value pair that has course_number for the name and 1057 for the value.

> **See Also:** "Rule Action Context" on page 5-8 for a scenario that uses such a name-value pair in an action context

Altering a Rule

You can use the ALTER_RULE procedure in the DBMS_RULE_ADM package to alter an existing **rule**. Specifically, you can use this procedure to do the following:

- Change a **rule condition**
- Change a rule **evaluation context**
- Remove a rule evaluation context
- Modify a name-value pair in a rule **action context**
- Add a name-value pair to a rule action context
- Remove a name-value pair from a rule action context
- Change the comment for a rule
- Remove the comment for a rule

The following sections contains examples for some of these alterations.

Changing a Rule Condition

You use the `condition` parameter in the `ALTER_RULE` procedure to change the condition of an existing rule. For example, suppose you want to change the condition of the rule created in "Creating a Rule" on page 14-4. The condition in the existing `hr_dml` rule evaluates to `TRUE` for any DML change to any object in the `hr` schema. If you want to exclude changes to the `employees` table in this schema, then you can alter the rule so that it evaluates to `FALSE` for DML changes to the `hr.employees` table, but continues to evaluate to `TRUE` for DML changes to any other table in this schema. The following procedure alters the rule in this way:

```
BEGIN
  DBMS_RULE_ADM.ALTER_RULE(
    rule_name          => 'strmadmin.hr_dml',
    condition          => ' :dml.get_object_owner() = ''HR'' AND NOT ' ||
                          ' :dml.get_object_name() = ''EMPLOYEES'' ',
    evaluation_context => NULL);
END;
/
```

Note:

- Changing the condition of a rule affects all **rule set**s that contain the rule.
- If you want to alter a rule but retain the rule action context, then specify `NULL` for `action_context` parameter in the `ALTER_RULE` procedure. `NULL` is the default value for the `action_context` parameter.

Modifying a Name-Value Pair in a Rule Action Context

To modify a name-value pair in a rule action context, you first remove the name-value pair from the rule action context and then add a different name-value pair to the rule action context.

This example modifies a name-value pair for rule `rule_dep_10` by first removing the name-value pair with the name `course_name` from the rule action context and then adding a different name-value pair back to the rule action context with the same name (`course_name`) but a different value. This name-value pair being modified was added to the rule in the example in "Creating a Rule with an Action Context" on page 14-6.

If an action context contains name-value pairs in addition to the name-value pair that you are modifying, then be cautious when you modify the action context so that you do not change or remove any of the other name-value pairs.

Complete the following steps to modify a name-value pair in an action context:

1. You can view the name-value pairs in the action context of a rule by performing the following query:

```
COLUMN ACTION_CONTEXT_NAME HEADING 'Action Context Name' FORMAT A25
COLUMN AC_VALUE_NUMBER HEADING 'Action Context Number Value' FORMAT 9999

SELECT
    AC.NVN_NAME ACTION_CONTEXT_NAME,
    AC.NVN_VALUE.ACCESSNUMBER() AC_VALUE_NUMBER
  FROM DBA_RULES R, TABLE(R.RULE_ACTION_CONTEXT.ACTX_LIST) AC
  WHERE RULE_NAME = 'RULE_DEP_10';
```

This query displays output similar to the following:

```
Action Context Name       Action Context Number Value
------------------------- --------------------------
course_number                                 1057
```

2. Modify the name-value pair. Make sure no other users are modifying the action context at the same time. This step first removes the name-value pair containing the name `course_number` from the action context for the `rule_dep_10` rule using the `REMOVE_PAIR` member procedure of the `RE$NV_LIST` type. Next, this step adds a name-value pair containing the new name-value pair to the rule action context using the `ADD_PAIR` member procedure of this type. In this case, the name is `course_number` and the value is `1108` for the added name-value pair.

   To preserve any existing name-value pairs in the rule action context, this example selects the rule action context into a variable before altering it:

```
DECLARE
  action_ctx       SYS.RE$NV_LIST;
  ac_name          VARCHAR2(30) := 'course_number';
BEGIN
  SELECT RULE_ACTION_CONTEXT
    INTO action_ctx
    FROM DBA_RULES R
    WHERE RULE_OWNER='STRMADMIN' AND RULE_NAME='RULE_DEP_10';
  action_ctx.REMOVE_PAIR(ac_name);
  action_ctx.ADD_PAIR(ac_name,
                ANYDATA.CONVERTNUMBER(1108));
  DBMS_RULE_ADM.ALTER_RULE(
    rule_name       =>  'strmadmin.rule_dep_10',
    action_context  => action_ctx);
END;
/
```

   To ensure that the name-value pair was altered properly, you can rerun the query in Step 1. The query should display output similar to the following:

```
Action Context Name       Action Context Number Value
------------------------- --------------------------
course_number                                 1108
```

Adding a Name-Value Pair to a Rule Action Context

You can preserve the existing name-value pairs in the action context by selecting the action context into a variable before adding a new pair using the `ADD_PAIR` member procedure of the `RE$NV_LIST` type. Make sure no other users are modifying the action context at the same time. The following example preserves the existing name-value pairs in the action context of the `rule_dep_10` rule and adds a new name-value pair with `dist_list` for the name and `admin_list` for the value:

```
DECLARE
  action_ctx       SYS.RE$NV_LIST;
  ac_name          VARCHAR2(30) := 'dist_list';
BEGIN
  action_ctx := SYS.RE$NV_LIST(SYS.RE$NV_ARRAY());
  SELECT RULE_ACTION_CONTEXT
    INTO action_ctx
    FROM DBA_RULES R
    WHERE RULE_OWNER='STRMADMIN' AND RULE_NAME='RULE_DEP_10';
```

```
  action_ctx.ADD_PAIR(ac_name,
                ANYDATA.CONVERTVARCHAR2('admin_list'));
  DBMS_RULE_ADM.ALTER_RULE(
    rule_name       =>  'strmadmin.rule_dep_10',
    action_context  => action_ctx);
END;
/
```

To make sure the name-value pair was added successfully, you can run the following query:

```
COLUMN ACTION_CONTEXT_NAME HEADING 'Action Context Name' FORMAT A25
COLUMN AC_VALUE_NUMBER HEADING 'Action Context|Number Value' FORMAT 9999
COLUMN AC_VALUE_VARCHAR2 HEADING 'Action Context|Text Value' FORMAT A25

SELECT
    AC.NVN_NAME ACTION_CONTEXT_NAME,
    AC.NVN_VALUE.ACCESSNUMBER() AC_VALUE_NUMBER,
    AC.NVN_VALUE.ACCESSVARCHAR2() AC_VALUE_VARCHAR2
  FROM DBA_RULES R, TABLE(R.RULE_ACTION_CONTEXT.ACTX_LIST) AC
  WHERE RULE_NAME = 'RULE_DEP_10';
```

This query should display output similar to the following:

```
                          Action Context Action Context
Action Context Name         Number Value Text Value
------------------------- -------------- -------------------------
course_number                       1088
dist_list                                admin_list
```

> **See Also:** "Rule Action Context" on page 5-8 for a scenario that uses similar name-value pairs in an action context

Removing a Name-Value Pair from a Rule Action Context

You remove a name-value pair in the action context of a rule using the `REMOVE_PAIR` member procedure of the `RE$NV_LIST` type. Make sure no other users are modifying the action context at the same time.

Removing a name-value pair means altering the action context of a rule. If an action context contains name-value pairs in addition to the name-value pair being removed, then be cautious when you modify the action context so that you do not change or remove any other name-value pairs.

This example assumes that the `rule_dep_10` rule has the following name-value pairs:

Name	Value
course_number	1088
dist_list	admin_list

See Also: You added these name-value pairs to the `rule_dep_10` rule if you completed the examples in the following sections:

- "Creating a Rule with an Action Context" on page 14-6
- "Modifying a Name-Value Pair in a Rule Action Context" on page 14-7
- "Adding a Name-Value Pair to a Rule Action Context" on page 14-8

This example preserves existing name-value pairs in the action context of the `rule_dep_10` rule that should not be removed by selecting the existing action context into a variable and then removing the name-value pair with `dist_list` for the name.

```
DECLARE
  action_ctx       SYS.RE$NV_LIST;
  ac_name          VARCHAR2(30) := 'dist_list';
BEGIN
  SELECT RULE_ACTION_CONTEXT
    INTO action_ctx
    FROM DBA_RULES R
    WHERE RULE_OWNER='STRMADMIN' AND RULE_NAME='RULE_DEP_10';
  action_ctx.REMOVE_PAIR(ac_name);
  DBMS_RULE_ADM.ALTER_RULE(
    rule_name       =>  'strmadmin.rule_dep_10',
    action_context  =>  action_ctx);
END;
/
```

To make sure the name-value pair was removed successfully without removing any other name-value pairs in the action context, you can run the following query:

```
COLUMN ACTION_CONTEXT_NAME HEADING 'Action Context Name' FORMAT A25
COLUMN AC_VALUE_NUMBER HEADING 'Action Context|Number Value' FORMAT 9999
COLUMN AC_VALUE_VARCHAR2 HEADING 'Action Context|Text Value' FORMAT A25

SELECT
    AC.NVN_NAME ACTION_CONTEXT_NAME,
    AC.NVN_VALUE.ACCESSNUMBER() AC_VALUE_NUMBER,
    AC.NVN_VALUE.ACCESSVARCHAR2() AC_VALUE_VARCHAR2
  FROM DBA_RULES R, TABLE(R.RULE_ACTION_CONTEXT.ACTX_LIST) AC
  WHERE RULE_NAME = 'RULE_DEP_10';
```

This query should display output similar to the following:

```
                        Action Context Action Context
Action Context Name       Number Value Text Value
----------------------- -------------- ------------------------
course_number                     1108
```

Modifying System-Created Rules

System-created **rule**s are rules created by running a procedure in the `DBMS_STREAMS_ADM` package. If you cannot create a rule with the exact **rule condition** you need using the `DBMS_STREAMS_ADM` package, then you can create a new rule with a condition based on a **system-created rule** by following these general steps:

1. Copy the rule condition of the system-created rule. You can view the rule condition of a system-created rule by querying the DBA_STREAMS_RULES data dictionary view.
2. Modify the condition.
3. Create a new rule with the modified condition.
4. Add the new rule to a **rule set** for a Streams **capture process**, **propagation**, **apply process**, or **messaging client**.
5. Remove the original rule if it is no longer needed using the REMOVE_RULE procedure in the DBMS_STREAMS_ADM package.

See Also:

- Chapter 7, "Rule-Based Transformations"
- Chapter 19, "Monitoring a Streams Environment" for more information about the data dictionary views related to Streams

Dropping a Rule

The following example runs the DROP_RULE procedure in the DBMS_RULE_ADM package to drop the hr_dml **rule** from the database:

```
BEGIN
  DBMS_RULE_ADM.DROP_RULE(
    rule_name => 'strmadmin.hr_dml',
    force     => false);
END;
/
```

In this example, the force parameter in the DROP_RULE procedure is set to false, which is the default setting. Therefore, the rule cannot be dropped if it is in one or more **rule set**s. If the force parameter is set to true, then the rule is dropped from the database and automatically removed from any rule sets that contain it.

Managing Privileges on Evaluation Contexts, Rule Sets, and Rules

This section provides instructions for completing the following tasks:

- Granting System Privileges on Evaluation Contexts, Rule Sets, and Rules
- Granting Object Privileges on an Evaluation Context, Rule Set, or Rule
- Revoking System Privileges on Evaluation Contexts, Rule Sets, and Rules
- Revoking Object Privileges on an Evaluation Context, Rule Set, or Rule

See Also:

- "Database Objects and Privileges Related to Rules" on page 5-13
- The GRANT_SYSTEM_PRIVILEGE and GRANT_OBJECT_PRIVILEGE procedures in the DBMS_RULE_ADM package in *Oracle Database PL/SQL Packages and Types Reference*

Granting System Privileges on Evaluation Contexts, Rule Sets, and Rules

You can use the GRANT_SYSTEM_PRIVILEGE procedure in the DBMS_RULE_ADM package to grant system privileges on **evaluation context**s, **rule set**s, and **rule**s to users and roles. These privileges enable a user to create, alter, execute, or drop these objects in the user's own schema or, if the "ANY" version of the privilege is granted, in any schema.

For example, to grant the hr user the privilege to create an evaluation context in the user's own schema, enter the following while connected as a user who can grant privileges and alter users:

```
BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => SYS.DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT_OBJ,
    grantee      => 'hr',
    grant_option => false);
END;
/
```

In this example, the grant_option parameter in the GRANT_SYSTEM_PRIVILEGE procedure is set to false, which is the default setting. Therefore, the hr user cannot grant the CREATE_EVALUATION_CONTEXT_OBJ system privilege to other users or roles. If the grant_option parameter were set to true, then the hr user could grant this system privilege to other users or roles.

Granting Object Privileges on an Evaluation Context, Rule Set, or Rule

You can use the GRANT_OBJECT_PRIVILEGE procedure in the DBMS_RULE_ADM package to grant object privileges on a specific **evaluation context**, **rule set**, or **rule**. These privileges enable a user to alter or execute the specified object.

For example, to grant the hr user the privilege to both alter and execute a rule set named hr_capture_rules in the strmadmin schema, enter the following:

```
BEGIN
  DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE(
    privilege    => SYS.DBMS_RULE_ADM.ALL_ON_RULE_SET,
    object_name  => 'strmadmin.hr_capture_rules',
    grantee      => 'hr',
    grant_option => false);
END;
/
```

In this example, the grant_option parameter in the GRANT_OBJECT_PRIVILEGE procedure is set to false, which is the default setting. Therefore, the hr user cannot grant the ALL_ON_RULE_SET object privilege for the specified rule set to other users or roles. If the grant_option parameter were set to true, then the hr user could grant this object privilege to other users or roles.

Revoking System Privileges on Evaluation Contexts, Rule Sets, and Rules

You can use the REVOKE_SYSTEM_PRIVILEGE procedure in the DBMS_RULE_ADM package to revoke system privileges on **evaluation context**s, **rule set**s, and **rule**s.

For example, to revoke from the hr user the privilege to create an evaluation context in the user's own schema, enter the following while connected as a user who can grant privileges and alter users:

```
BEGIN
  DBMS_RULE_ADM.REVOKE_SYSTEM_PRIVILEGE(
    privilege    => SYS.DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT_OBJ,
    revokee      => 'hr');
END;
/
```

Revoking Object Privileges on an Evaluation Context, Rule Set, or Rule

You can use the REVOKE_OBJECT_PRIVILEGE procedure in the DBMS_RULE_ADM package to revoke object privileges on a specific **evaluation context**, **rule set**, or **rule**.

For example, to revoke from the hr user the privilege to both alter and execute a rule set named hr_capture_rules in the strmadmin schema, enter the following:

```
BEGIN
  DBMS_RULE_ADM.REVOKE_OBJECT_PRIVILEGE(
    privilege    => SYS.DBMS_RULE_ADM.ALL_ON_RULE_SET,
    object_name  => 'strmadmin.hr_capture_rules',
    revokee      => 'hr');
END;
/
```

15

Managing Rule-Based Transformations

In Streams, a rule-based transformation is any modification to a **message** that results when a **rule** in a **positive rule set** evaluates to `TRUE`. There are two types of rule-based transformations: declarative and custom. This chapter describes managing each type of rule-based transformation.

- Managing Declarative Rule-Based Transformations
- Managing Custom Rule-Based Transformations

Note: A transformation specified for a rule is performed only if the rule is in a positive rule set. If the rule is in the **negative rule set** for a **capture process**, **propagation**, **apply process**, or **messaging client**, then these **Streams client**s ignore the rule-based transformation.

See Also: Chapter 7, "Rule-Based Transformations" for conceptual information about each type of rule-based transformation

Managing Declarative Rule-Based Transformations

You can use the following procedures in the `DBMS_STREAMS_ADM` package to manage **declarative rule-based transformation**s: `ADD_COLUMN`, `DELETE_COLUMN`, `RENAME_COLUMN`, `RENAME_SCHEMA`, and `RENAME_TABLE`.

This section provides instructions for completing the following tasks:

- Adding Declarative Rule-Based Transformations
- Removing Declarative Rule-Based Transformations

Adding Declarative Rule-Based Transformations

The following sections contain examples that add **declarative rule-based transformation**s to **rule**s.

Adding a Declarative Rule-Based Transformation that Renames a Table

Use the `RENAME_TABLE` procedure in the `DBMS_STREAMS_ADM` package to add a declarative rule-based transformation that renames a table in a row LCR. For example, the following procedure adds a declarative rule-based transformation to the `jobs12` rule in the `strmadmin` schema:

```
BEGIN
  DBMS_STREAMS_ADM.RENAME_TABLE(
    rule_name       => 'strmadmin.jobs12',
    from_table_name => 'hr.jobs',
    to_table_name   => 'hr.assignments',
    step_number     => 0,
    operation       => 'ADD');
END;
/
```

The declarative rule-based transformation added by this procedure renames the table `hr.jobs` to `hr.assignments` in a row LCR when the rule `jobs12` evaluates to `TRUE` for the row LCR. If more than one declarative rule-based transformation is specified for the `jobs12` rule, then this transformation follows default transformation ordering because the `step_number` parameter is set to `0` (zero). In addition, the `operation` parameter is set to `ADD` to indicate that the transformation is being added to the rule, not removed from it.

The `RENAME_TABLE` procedure can also add a transformation that renames the schema in addition to the table. For example, in the previous example, to specify that the schema should be renamed to `oe`, specify `oe.assignments` for the `to_table_name` parameter.

Adding a Declarative Rule-Based Transformation that Adds a Column

Use the `ADD_COLUMN` procedure in the `DBMS_STREAMS_ADM` package to add a declarative rule-based transformation that adds a column to a row in a row LCR. For example, the following procedure adds a declarative rule-based transformation to the `employees35` rule in the `strmadmin` schema:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_COLUMN(
    rule_name    => 'employees35',
    table_name   => 'hr.employees',
    column_name  => 'birth_date',
    column_value => ANYDATA.ConvertDate(NULL),
    value_type   => 'NEW',
    step_number  => 0,
    operation    => 'ADD');
END;
/
```

The declarative rule-based transformation added by this procedure adds a `birth_date` column of datatype `DATE` to an `hr.employees` table row in a row LCR when the rule `employees35` evaluates to `TRUE` for the row LCR.

Notice that the `ANYDATA.ConvertDate` function specifies the column type and the column value. In this example, the added column value is `NULL`, but a valid date can also be specified. Use the appropriate `AnyData` function for the column being added. For example, if the datatype of the column being added is `NUMBER`, then use the `ANYDATA.ConvertNumber` function.

The `value_type` parameter is set to `NEW` to indicate that the column is added to the new values in a row LCR. You can also specify `OLD` to add the column to the old values.

If more than one declarative rule-based transformation is specified for the `employees35` rule, then the transformation follows default transformation ordering because the `step_number` parameter is set to `0` (zero). In addition, the `operation` parameter is set to `ADD` to indicate that the transformation is being added, not removed.

Note: The `ADD_COLUMN` procedure is overloaded. A `column_function` parameter can specify that the current system date or timestamp is the value for the added column. The `column_value` and `column_function` parameters are mutually exclusive.

See Also: *Oracle Database PL/SQL Packages and Types Reference* for more information about `AnyData` type functions

Overwriting an Existing Declarative Rule-Based Transformation

When the `operation` parameter is set to `ADD` in a procedure that adds a **declarative rule-based transformation**, an existing declarative rule-based transformation is overwritten if the parameters in the following list match the existing transformation parameters:

- `ADD_COLUMN` procedure: `rule_name`, `table_name`, `column_name`, and `step_number` parameters
- `DELETE_COLUMN` procedure: `rule_name`, `table_name`, `column_name`, and `step_number` parameters
- `RENAME_COLUMN` procedure: `rule_name`, `table_name`, `from_column_name`, and `step_number` parameters
- `RENAME_SCHEMA` procedure: `rule_name`, `from_schema_name`, and `step_number` parameters
- `RENAME_TABLE` procedure: `rule_name`, `from_table_name`, and `step_number` parameters

For example, suppose an existing declarative rule-based transformation was creating by running the following procedure:

```
BEGIN
  DBMS_STREAMS_ADM.RENAME_COLUMN(
    rule_name         => 'departments33',
    table_name        => 'hr.departments',
    from_column_name  => 'manager_id',
    to_column_name    => 'lead_id',
    value_type        => 'NEW',
    step_number       => 0,
    operation         => 'ADD');
END;
/
```

Running the following procedure overwrites this existing declarative rule-based transformation:

```
BEGIN
  DBMS_STREAMS_ADM.RENAME_COLUMN(
    rule_name         => 'departments33',
    table_name        => 'hr.departments',
    from_column_name  => 'manager_id',
    to_column_name    => 'lead_id',
    value_type        => '*',
    step_number       => 0,
    operation         => 'ADD');
END;
/
```

In this case, the `value_type` parameter in the declarative rule-based transformation was changed from `NEW` to `*`. That is, in the original transformation, only new values were renamed in row LCRs, but, in the new transformation, both old and new values are renamed in row LCRs.

Removing Declarative Rule-Based Transformations

To remove a **declarative rule-based transformation** from a **rule**, use the same procedure used to add the transformation, but specify `REMOVE` for the `operation` parameter. For example, to remove the transformation added in "Adding a Declarative Rule-Based Transformation that Renames a Table" on page 15-1, run the following procedure:

```
BEGIN
  DBMS_STREAMS_ADM.RENAME_TABLE(
    rule_name       => 'strmadmin.jobs12',
    from_table_name => 'hr.jobs',
    to_table_name   => 'hr.assignments',
    step_number     => 0,
    operation       => 'REMOVE');
END;
/
```

When the `operation` parameter is set to `REMOVE` in any of the declarative transformation procedures listed in "Managing Declarative Rule-Based Transformations" on page 15-1, the other parameters in the procedure are optional, excluding the `rule_name` parameter. If these optional parameters are set to `NULL`, then they become wildcards.

The `RENAME_TABLE` procedure in the previous example behaves in the following way when one or more of the optional parameters are set to `NULL`:

from_table_ name Parameter	to_table_name Parameter	step_number Parameter	Result
`NULL`	`NULL`	`NULL`	Remove all rename table transformations for the specified rule
non-`NULL`	`NULL`	`NULL`	Remove all rename table transformations with the specified `from_table_name` for the specified rule

from_table_name Parameter	to_table_name Parameter	step_number Parameter	Result
NULL	non-NULL	NULL	Remove all rename table transformations with the specified to_table_name for the specified rule
NULL	NULL	non-NULL	Remove all rename table transformations with the specified step_number for the specified rule
non-NULL	non-NULL	NULL	Remove all rename table transformations with the specified from_table_name and to_table_name for the specified rule
NULL	non-NULL	non-NULL	Remove all rename table transformations with the specified to_table_name and step_number for the specified rule
non-NULL	NULL	non-NULL	Remove all rename table transformations with the specified from_table_name and step_number for the specified rule

The other declarative transformation procedures work in a similar way when optional parameters are set to NULL and the operation parameter is set to REMOVE.

Managing Custom Rule-Based Transformations

Use the SET_RULE_TRANSFORM_FUNCTION procedure in the DBMS_STREAMS_ADM package to set or unset a **custom rule-based transformation** for a **rule**. This procedure modifies the rule **action context** to specify the custom rule-based transformation.

This section provides instructions for completing the following tasks:

- Creating a Custom Rule-Based Transformation
- Altering a Custom Rule-Based Transformation
- Unsetting a Custom Rule-Based Transformation

Attention: Do not modify LONG, LONG RAW, or LOB column data in an LCR with a custom rule-based transformation.

Note:

- There is no automatic locking mechanism for a rule action context. Therefore, make sure an action context is not updated by two or more sessions at the same time.
- When you perform custom rule-based transformations on DDL LCRs, you probably need to modify the DDL text in the DDL LCR to match any other modification. For example, if the transformation changes the name of a table in the DDL LCR, then the transformation should change the table name in the DDL text in the same way.

Creating a Custom Rule-Based Transformation

A **custom rule-based transformation** function always operates on one **message**, but it can return one message or many messages. A custom rule-based transformation function that returns one message is a one-to-one transformation function. A one-to-one transformation function must have the following signature:

```
FUNCTION user_function (
   parameter_name   IN  ANYDATA)
RETURN ANYDATA;
```

Here, *user_function* stands for the name of the function and *parameter_name* stands for the name of the parameter passed to the function. The parameter passed to the function is an ANYDATA encapsulation of a message, and the function must return an ANYDATA encapsulation of a message.

A custom rule-based transformation function that can return more than one message is a one-to-many transformation function. A one-to-many transformation function must have the following signature:

```
FUNCTION user_function (
   parameter_name   IN  ANYDATA)
RETURN STREAMS$_ANYDATA_ARRAY;
```

Here, *user_function* stands for the name of the function and *parameter_name* stands for the name of the parameter passed to the function. The parameter passed to the function is an ANYDATA encapsulation of a message, and the function must return an array that contains zero or more ANYDATA encapsulations of a message. If the array contains zero ANYDATA encapsulations of a message, then the original message is discarded. One-to-many transformation functions are supported only for Streams **capture process**es.

The STREAMS$_ANYDATA_ARRAY type is an Oracle-supplied type that has the following definition:

```
CREATE OR REPLACE TYPE SYS.STREAMS$_ANYDATA_ARRAY
   AS VARRAY(2147483647) of SYS.ANYDATA
/
```

The following steps outline the general procedure for creating a custom rule-based transformation that uses a one-to-one function:

1. Create a PL/SQL function that performs the transformation.

> **Caution:** Make sure the transformation function is deterministic. A deterministic function always returns the same value for any given set of input argument values, now and in the future. Also, make sure the transformation function does not raise any exceptions. Exceptions can cause a capture process, **propagation**, or **apply process** to become disabled, and you will need to correct the transformation function before the capture process, propagation, or apply process can proceed. Exceptions raised by a custom rule-based transformation for a **messaging client** can prevent the messaging client from dequeuing messages.

The following example creates a function called `executive_to_management` in the `hr` schema that changes the value in the `department_name` column of the `departments` table from `Executive` to `Management`. Such a transformation might be necessary if one branch in a company uses a different name for this department.

```
CONNECT hr/hr

CREATE OR REPLACE FUNCTION hr.executive_to_management(in_any IN ANYDATA)
RETURN ANYDATA
IS
  lcr SYS.LCR$_ROW_RECORD;
  rc  NUMBER;
  ob_owner VARCHAR2(30);
  ob_name VARCHAR2(30);
  dep_value_anydata ANYDATA;
  dep_value_varchar2 VARCHAR2(30);
BEGIN
  -- Get the type of object
  -- Check if the object type is SYS.LCR$_ROW_RECORD
  IF in_any.GETTYPENAME='SYS.LCR$_ROW_RECORD' THEN
    -- Put the row LCR into lcr
    rc := in_any.GETOBJECT(lcr);
    -- Get the object owner and name
    ob_owner := lcr.GET_OBJECT_OWNER();
    ob_name := lcr.GET_OBJECT_NAME();
    -- Check for the hr.departments table
    IF ob_owner = 'HR' AND ob_name = 'DEPARTMENTS' THEN
      -- Get the old value of the department_name column in the LCR
      dep_value_anydata := lcr.GET_VALUE('old','DEPARTMENT_NAME');
      IF dep_value_anydata IS NOT NULL THEN
        -- Put the column value into dep_value_varchar2
        rc := dep_value_anydata.GETVARCHAR2(dep_value_varchar2);
        -- Change a value of Executive in the column to Management
        IF (dep_value_varchar2 = 'Executive') THEN
          lcr.SET_VALUE('OLD','DEPARTMENT_NAME',
            ANYDATA.CONVERTVARCHAR2('Management'));
        END IF;
      END IF;
      -- Get the new value of the department_name column in the LCR
      dep_value_anydata := lcr.GET_VALUE('new', 'DEPARTMENT_NAME', 'n');
      IF dep_value_anydata IS NOT NULL THEN
        -- Put the column value into dep_value_varchar2
        rc := dep_value_anydata.GETVARCHAR2(dep_value_varchar2);
        -- Change a value of Executive in the column to Management
        IF (dep_value_varchar2 = 'Executive') THEN
          lcr.SET_VALUE('new','DEPARTMENT_NAME',
            ANYDATA.CONVERTVARCHAR2('Management'));
        END IF;
      END IF;
    END IF;
    RETURN ANYDATA.CONVERTOBJECT(lcr);
  END IF;
RETURN in_any;
END;
/
```

2. Grant the Streams administrator `EXECUTE` privilege on the `hr.executive_to_management` function.

```
GRANT EXECUTE ON hr.executive_to_management TO strmadmin;
```

3. Create **subset rule**s for DML operations on the `hr.departments` table. The subset rules will use the transformation created in Step 1.

   Subset rules are not required to use custom rule-based transformations. This example uses subset rules to illustrate an **action context** with more than one name-value pair. This example creates subset rules for an apply process on a database named `dbs1.net`. These rules evaluate to `TRUE` when an LCR contains a DML change to a row with a `location_id` of `1700` in the `hr.departments` table. This example assumes that an `ANYDATA` queue named `strm01_queue` already exists in the database.

   To create these rules, connect as the Streams administrator and run the following `ADD_SUBSET_RULES` procedure:

```
CONNECT strmadmin/strmadminpw

BEGIN
  DBMS_STREAMS_ADM.ADD_SUBSET_RULES(
    table_name               =>  'hr.departments',
    dml_condition            =>  'location_id=1700',
    streams_type             =>  'apply',
    streams_name             =>  'strm01_apply',
    queue_name               =>  'strm01_queue',
    include_tagged_lcr       =>  false,
    source_database          =>  'dbs1.net');
END;
/
```

   Note:

   - To create the rule and the **rule set**, the Streams administrator must have `CREATE_RULE_SET_OBJ` (or `CREATE_ANYRULE_SET_OBJ`) and `CREATE_RULE_OBJ` (or `CREATE_ANY_RULE_OBJ`) system privileges. You grant these privileges using the `GRANT_SYSTEM_PRIVILEGE` procedure in the `DBMS_RULE_ADM` package.
   - This example creates the rule using the `DBMS_STREAMS_ADM` package. Alternatively, you can create a rule, add it to a rule set, and specify a custom rule-based transformation using the `DBMS_RULE_ADM` package. *Oracle Streams Replication Administrator's Guide* contains an example of this procedure.
   - The `ADD_SUBSET_RULES` procedure adds the subset rules to the **positive rule set** for the apply process.

4. Determine the names of the system-created **rule**s by running the following query:

```
SELECT RULE_NAME, SUBSETTING_OPERATION FROM DBA_STREAMS_RULES
  WHERE OBJECT_NAME='DEPARTMENTS' AND DML_CONDITION='location_id=1700';
```

   This query displays output similar to the following:

```
RULE_NAME                      SUBSET
------------------------------ ------
DEPARTMENTS5                   INSERT
DEPARTMENTS6                   UPDATE
DEPARTMENTS7                   DELETE
```

> **Note:** You can also obtain this information using the OUT parameters when you run ADD_SUBSET_RULES.

Because these are subset rules, two of them contain a non-NULL action context that performs an internal transformation:

- The rule with a subsetting condition of INSERT contains an internal transformation that converts updates into inserts if the update changes the value of the location_id column to 1700 from some other value. The internal transformation does not affect inserts.
- The rule with a subsetting condition of DELETE contains an internal transformation that converts updates into deletes if the update changes the value of the location_id column from 1700 to a different value. The internal transformation does not affect deletes.

In this example, you can confirm that the rules DEPARTMENTS5 and DEPARTMENTS7 have a non-NULL action context, and that the rule DEPARTMENTS6 has a NULL action context, by running the following query:

```
COLUMN RULE_NAME HEADING 'Rule Name' FORMAT A13
COLUMN ACTION_CONTEXT_NAME HEADING 'Action Context Name' FORMAT A27
COLUMN ACTION_CONTEXT_VALUE HEADING 'Action Context Value' FORMAT A30

SELECT
    RULE_NAME,
    AC.NVN_NAME ACTION_CONTEXT_NAME,
    AC.NVN_VALUE.ACCESSVARCHAR2() ACTION_CONTEXT_VALUE
  FROM DBA_RULES R, TABLE(R.RULE_ACTION_CONTEXT.ACTX_LIST) AC
  WHERE RULE_NAME IN ('DEPARTMENTS5','DEPARTMENTS6','DEPARTMENTS7');
```

This query displays output similar to the following:

```
Rule Name     Action Context Name         Action Context Value
------------- --------------------------- ------------------------------
DEPARTMENTS5  STREAMS$_ROW_SUBSET         INSERT
DEPARTMENTS7  STREAMS$_ROW_SUBSET         DELETE
```

The DEPARTMENTS6 rule does not appear in the output because its action context is NULL.

5. Set the custom rule-based transformation for each subset rule by running the SET_RULE_TRANSFORM_FUNCTION procedure. This step runs this procedure for each rule and specifies hr.executive_to_management as the transformation function. Make sure no other users are modifying the action context at the same time.

```
BEGIN
  DBMS_STREAMS_ADM.SET_RULE_TRANSFORM_FUNCTION(
    rule_name           => 'departments5',
    transform_function  => 'hr.executive_to_management');
  DBMS_STREAMS_ADM.SET_RULE_TRANSFORM_FUNCTION(
    rule_name           => 'departments6',
    transform_function  => 'hr.executive_to_management');
  DBMS_STREAMS_ADM.SET_RULE_TRANSFORM_FUNCTION(
    rule_name           => 'departments7',
    transform_function  => 'hr.executive_to_management');
END;
```

```
/
```

Specifically, this procedure adds a name-value pair to each rule action context that specifies the name STREAMS$_TRANSFORM_FUNCTION and a value that is an ANYDATA instance containing the name of the PL/SQL function that performs the transformation. In this case, the transformation function is hr.executive_to_management.

> **Note:** The SET_RULE_TRANSFORM_FUNCTION does not verify that the specified transformation function exists. If the function does not exist, then an error is raised when a Streams process or job tries to invoke the transformation function.

Now, if you run the query that displays the name-value pairs in the action context for these rules, each rule, including the DEPARTMENTS6 rule, shows the name-value pair for the custom rule-based transformation:

```
SELECT
    RULE_NAME,
    AC.NVN_NAME ACTION_CONTEXT_NAME,
    AC.NVN_VALUE.ACCESSVARCHAR2() ACTION_CONTEXT_VALUE
  FROM DBA_RULES R, TABLE(R.RULE_ACTION_CONTEXT.ACTX_LIST) AC
  WHERE RULE_NAME IN ('DEPARTMENTS5','DEPARTMENTS6','DEPARTMENTS7');
```

This query displays output similar to the following:

```
Rule Name     Action Context Name         Action Context Value
------------- --------------------------- ------------------------------
DEPARTMENTS51 STREAMS$_ROW_SUBSET         INSERT
DEPARTMENTS51 STREAMS$_TRANSFORM_FUNCTION "HR"."EXECUTIVE_TO_MANAGEMENT"
DEPARTMENTS52 STREAMS$_TRANSFORM_FUNCTION "HR"."EXECUTIVE_TO_MANAGEMENT"
DEPARTMENTS53 STREAMS$_ROW_SUBSET         DELETE
DEPARTMENTS53 STREAMS$_TRANSFORM_FUNCTION "HR"."EXECUTIVE_TO_MANAGEMENT"
```

You can also view transformation functions using the DBA_STREAMS_TRANSFORM_FUNCTION data dictionary view.

> **See Also:** *Oracle Database PL/SQL Packages and Types Reference* for more information about the SET_RULE_TRANSFORM_FUNCTION and the rule types used in this example

Altering a Custom Rule-Based Transformation

To alter a **custom rule-based transformation**, you can either edit the transformation function or run the SET_RULE_TRANSFORM_FUNCTION procedure to specify a different transformation function. This example runs the SET_RULE_TRANSFORM_FUNCTION procedure to specify a different transformation function. The SET_RULE_TRANSFORM_FUNCTION procedure modifies the **action context** of a specified **rule** to run a different transformation function. If you edit the transformation function itself, then you do not need to run this procedure.

This example alters a custom rule-based transformation for rule DEPARTMENTS5 by changing the transformation function from hr.execute_to_management to hr.executive_to_lead. The hr.execute_to_management rule-based transformation was added to the DEPARTMENTS5 rule in the example in "Creating a Custom Rule-Based Transformation" on page 15-6.

In Streams, **subset rule**s use name-value pairs in an action context to perform internal transformations that convert UPDATE operations into INSERT and DELETE operations in some situations. Such a conversion is called a **row migration**. The SET_RULE_TRANSFORM_FUNCTION procedure preserves the name-value pairs that perform row migrations.

> **See Also:** "Row Migration and Subset Rules" on page 6-20 for more information about row migration

Complete the following steps to alter a custom rule-based transformation:

1. You can view all of the name-value pairs in the action context of a rule by performing the following query:

```
COLUMN ACTION_CONTEXT_NAME HEADING 'Action Context Name' FORMAT A30
COLUMN ACTION_CONTEXT_VALUE HEADING 'Action Context Value' FORMAT A30

SELECT
    AC.NVN_NAME ACTION_CONTEXT_NAME,
    AC.NVN_VALUE.ACCESSVARCHAR2() ACTION_CONTEXT_VALUE
  FROM DBA_RULES R, TABLE(R.RULE_ACTION_CONTEXT.ACTX_LIST) AC
  WHERE RULE_NAME = 'DEPARTMENTS5';
```

   This query displays output similar to the following:

```
Action Context Name            Action Context Value
------------------------------ ------------------------------
STREAMS$_ROW_SUBSET            INSERT
STREAMS$_TRANSFORM_FUNCTION    "HR"."EXECUTIVE_TO_MANAGEMENT"
```

2. Run the SET_RULE_TRANSFORM_FUNCTION procedure to set the transformation function to executive_to_lead for the DEPARTMENTS5 rule. In this example, it is assumed that the new transformation function is hr.executive_to_lead and that the strmadmin user has EXECUTE privilege on it.

```
BEGIN
  DBMS_STREAMS_ADM.SET_RULE_TRANSFORM_FUNCTION(
    rule_name           => 'departments5',
    transform_function  => 'hr.executive_to_lead');
END;
/
```

   To ensure that the transformation function was altered properly, you can rerun the query in Step 1. You should alter the action context for the DEPARTMENTS6 and DEPARTMENTS7 rules in a similar way to keep the three subset rules consistent.

> **Note:**
>
> - The SET_RULE_TRANSFORM_FUNCTION does not verify that the specified transformation function exists. If the function does not exist, then an error is raised when a Streams process or job tries to invoke the transformation function.
> - If a custom rule-based transformation function is modified at the same time that a **Streams client** tries to access it, then an error might be raised.

Unsetting a Custom Rule-Based Transformation

To unset a **custom rule-based transformation** from a **rule**, run the SET_RULE_TRANSFORM_FUNCTION procedure and specify NULL for the transformation function. Specifying NULL unsets the name-value pair that specifies the custom rule-based transformation in the rule **action context**. This example unsets a custom rule-based transformation for rule DEPARTMENTS5. This transformation was added to the DEPARTMENTS5 rule in the example in "Creating a Custom Rule-Based Transformation" on page 15-6.

In Streams, **subset rule**s use name-value pairs in an action context to perform internal transformations that convert UPDATE operations into INSERT and DELETE operations in some situations. Such a conversion is called a **row migration**. The SET_RULE_TRANSFORM_FUNCTION procedure preserves the name-value pairs that perform row migrations.

> **See Also:** "Row Migration and Subset Rules" on page 6-20 for more information about row migration

Run the following procedure to unset the custom rule-based transformation for rule DEPARTMENTS5:

```
BEGIN
  DBMS_STREAMS_ADM.SET_RULE_TRANSFORM_FUNCTION(
    rule_name           => 'departments5',
    transform_function  => NULL);
END;
/
```

To ensure that the transformation function was unset, you can run the query in Step 1 on page 15-11. You should alter the action context for the DEPARTMENTS6 and DEPARTMENTS7 rules in a similar way to keep the three subset rules consistent.

> **See Also:** "Row Migration and Subset Rules" on page 6-20 for more information about row migration

16

Using Information Provisioning

This chapter describes how to use information provisioning. This chapter includes an example that creates a **tablespace repository**, examples that transfer tablespaces between databases, and an example that uses a **file group repository** to store different versions of files.

This chapter contains these topics:

- Using a Tablespace Repository
- Using a File Group Repository

See Also: Chapter 8, "Information Provisioning"

Using a Tablespace Repository

The following procedures in the `DBMS_STREAMS_TABLESPACE_ADM` package can create a **tablespace repository**, add versioned tablespace sets to a tablespace repository, and copy versioned tablespace sets from a tablespace repository:

- `ATTACH_TABLESPACES`: This procedure copies a **version** of a tablespace set from a tablespace repository and attaches the tablespaces to a database.
- `CLONE_TABLESPACES`: This procedure adds a new version of a tablespace set to a tablespace repository by copying the tablespace set from a database. The tablespaces in the tablespace set remain part of the database from which they were copied.
- `DETACH_TABLESPACES`: This procedure adds a new version of a tablespace set to a tablespace repository by moving the tablespace set from a database to the repository. The tablespaces in the tablespace set are dropped from the database from which they were copied.

This section illustrates how to use a tablespace repository with an example scenario. In the scenario, the goal is to run quarterly reports on the sales tablespaces (`sales_tbs1` and `sales_tbs2`). Sales are recorded in these tablespaces in the `inst1.net` database. The example clones the tablespaces quarterly and stores a new version of the tablespaces in the tablespace repository. The tablespace repository also resides in the `inst1.net` database. When a specific version of the tablespace set is required to run reports at a reporting database, it is copied from the tablespace repository and attached to the reporting database.

In this example scenario, the following databases are the reporting databases:

- The reporting database `inst2.net` shares a file system with the `inst1.net` database. Also, the reports that are run on `inst2.net` might make changes to the tablespace. Therefore, the tablespaces are made read/write at `inst2.net`, and, when the reports are complete, a new version of the tablespace files is stored in a separate directory from the original version of the tablespace files.
- The reporting system `inst3.net` does not share a file system with the `inst1.net` database. The reports that are run on `inst3.net` do not make any changes to the tablespace. Therefore, the tablespaces remain read-only at `inst3.net`, and, when the reports are complete, the original version of the tablespace files remains in a single directory.

The following sections describe how to create and populate the tablespace repository and how to use the tablespace repository to run reports at the other databases:

- Creating and Populating a Tablespace Repository
- Using a Tablespace Repository for Remote Reporting with a Shared File System
- Using a Tablespace Repository for Remote Reporting Without a Shared File System

These examples must be run by an administrative user with the necessary privileges to run the procedures listed previously.

> **See Also:** *Oracle Database PL/SQL Packages and Types Reference* for more information about these procedures and the privileges required to run them

Creating and Populating a Tablespace Repository

This example creates a tablespaces repository and adds a new **version** of a tablespace set to the repository after each quarter. The tablespace set consists of the sales tablespaces for a business: `sales_tbs1` and `sales_tbs2`.

Figure 16–1 provides an overview of the **tablespace repository** created in this example:

Figure 16–1 Example Tablespace Repository

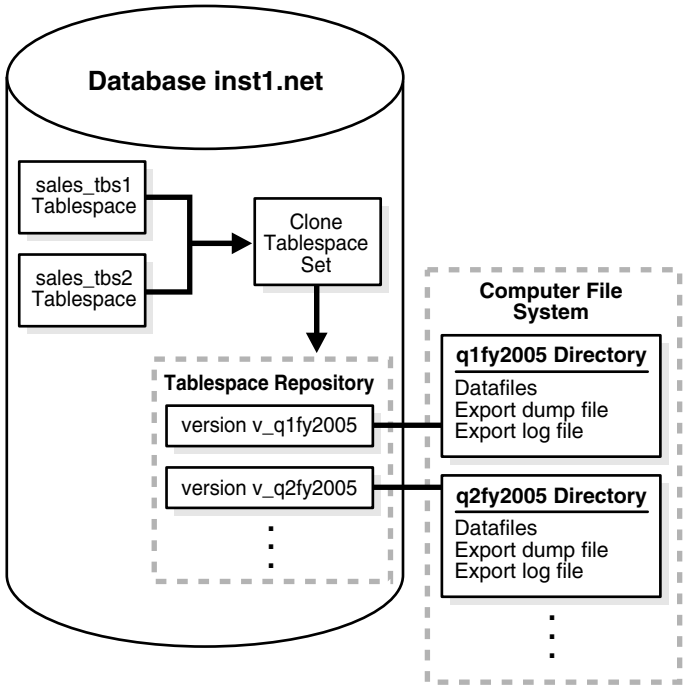


Table 16–1 shows the tablespace set versions created in this example, their directory objects, and the corresponding file system directory for each directory object.

Table 16–1 Versions in the Tablespace Repository

Version	Directory Object	Corresponding File System Directory
v_q1fy2005	q1fy2005	/home/sales/q1fy2005
v_q2fy2005	q2fy2005	/home/sales/q2fy2005

This example makes the following assumptions:

- The `inst1.net` database exists.
- The `sales_tbs1` and `sales_tbs2` tablespaces exist in the `inst1.net` database.

The following steps create and populate a tablespace repository:

1. Connect as an administrative user to the database where the sales tablespaces are modified with new sales data:

   ```
   CONNECT strmadmin/strmadminpw@inst1.net
   ```

   The administrative user must have the necessary privileges to run the procedures in the `DBMS_STREAMS_TABLESPACE_ADM` package and must have the necessary privileges to create directory objects.

2. Create a directory object for the first quarter in fiscal year 2005 on `inst1.net`:

   ```
   CREATE OR REPLACE DIRECTORY q1fy2005 AS '/home/sales/q1fy2005';
   ```

   The specified file system directory must exist when you create the directory object.

3. Create a directory object that corresponds to the directory that contains the datafiles for the tablespaces in the `inst1.net` database. For example, if the datafiles for the tablespaces are in the `/orc/inst1/dbs` directory, then create a directory object that corresponds to this directory:

```
CREATE OR REPLACE DIRECTORY dbfiles_inst1 AS '/orc/inst1/dbs';
```

4. Clone the tablespace set and add the first version of the tablespace set to the tablespace repository:

```
DECLARE
  tbs_set DBMS_STREAMS_TABLESPACE_ADM.TABLESPACE_SET;
BEGIN
  tbs_set(1) := 'sales_tbs1';
  tbs_set(2) := 'sales_tbs2';
  DBMS_STREAMS_TABLESPACE_ADM.CLONE_TABLESPACES(
    tablespace_names            => tbs_set,
    tablespace_directory_object => 'q1fy2005',
    file_group_name             => 'strmadmin.sales',
    version_name                => 'v_q1fy2005');
END;
/
```

   The `sales` **file group** is created automatically if it does not exist.

5. When the second quarter in fiscal year 2005 is complete, create a directory object for the second quarter in fiscal year 2005:

```
CREATE OR REPLACE DIRECTORY q2fy2005 AS '/home/sales/q2fy2005';
```

   The specified file system directory must exist when you create the directory object.

6. Clone the tablespace set and add the next version of the tablespace set to the tablespace repository at the `inst1.net` database:

```
DECLARE
  tbs_set DBMS_STREAMS_TABLESPACE_ADM.TABLESPACE_SET;
BEGIN
  tbs_set(1) := 'sales_tbs1';
  tbs_set(2) := 'sales_tbs2';
  DBMS_STREAMS_TABLESPACE_ADM.CLONE_TABLESPACES(
    tablespace_names            => tbs_set,
    tablespace_directory_object => 'q2fy2005',
    file_group_name             => 'strmadmin.sales',
    version_name                => 'v_q2fy2005');
END;
/
```

Steps 5 and 6 can be repeated whenever a quarter ends to store a version of the tablespace set for each quarter. Each time, create a new directory object to store the tablespace files for the quarter, and specify a unique version name for the quarter.

Using a Tablespace Repository for Remote Reporting with a Shared File System

This example runs reports at `inst2.net` on specific versions of the sales tablespaces stored in a **tablespace repository** at `inst1.net`. These two databases share a file system, and the reports that are run on `inst2.net` might make changes to the tablespace. Therefore, the tablespaces are made read/write at `inst2.net`. When the reports are complete, a new **version** of the tablespace files is stored in a separate directory from the original version of the tablespace files.

Figure 16–2 provides an overview of how tablespaces in a tablespace repository are attached to a different database in this example:

Figure 16–2 Attaching Tablespaces with a Shared File System

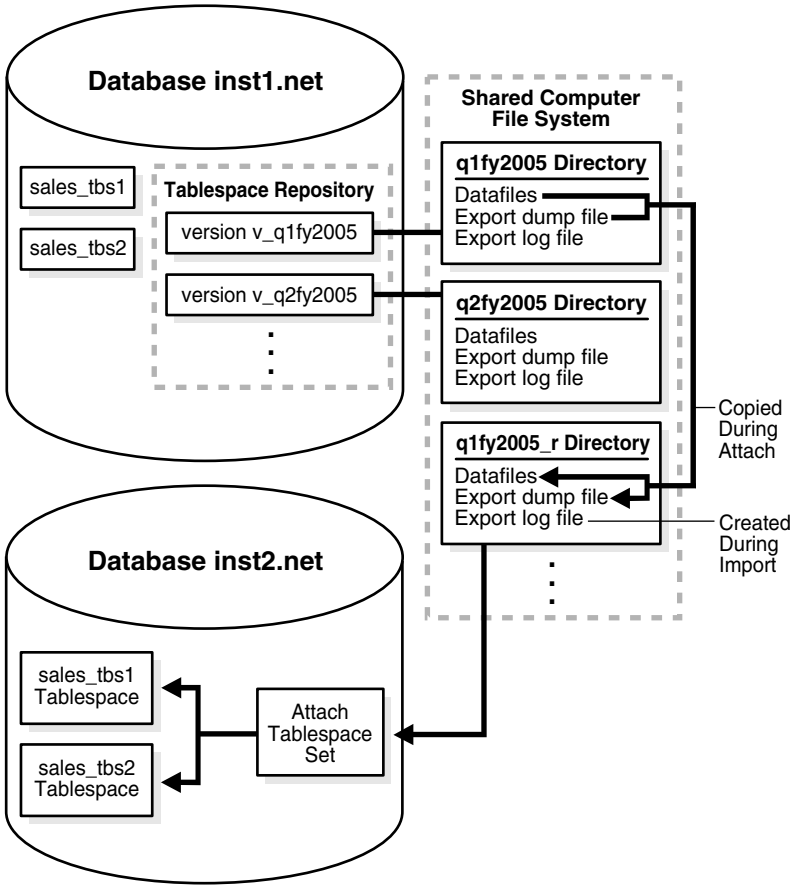


Figure 16–3 provides an overview of how tablespaces are detached and placed in a tablespace repository in this example:

Figure 16–3 Detaching Tablespaces with a Shared File System

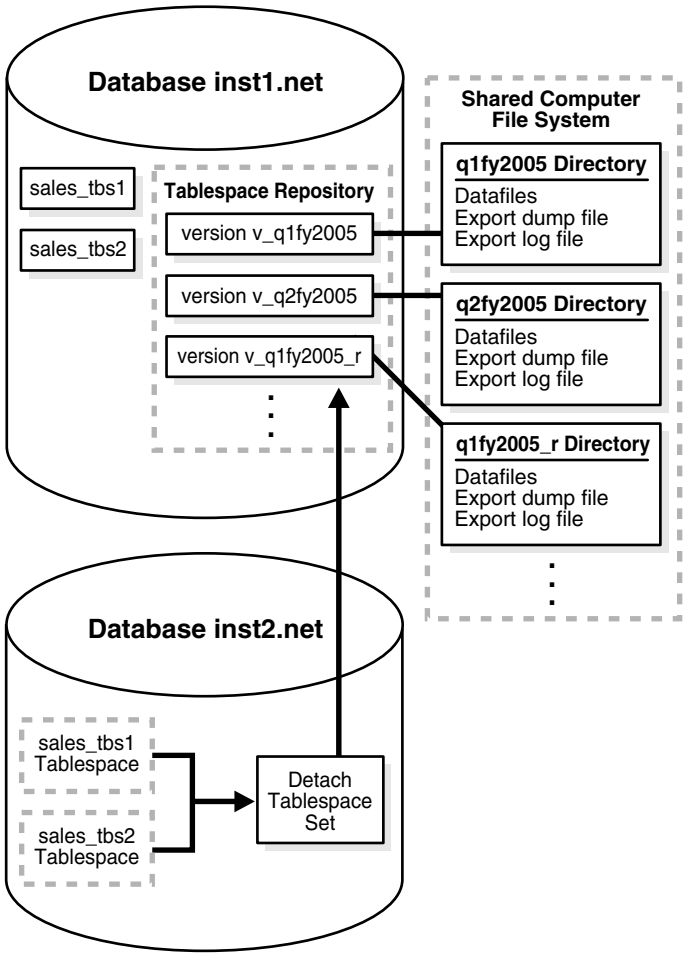


Table 16–2 shows the tablespace set versions in the tablespace repository when this example is complete. It shows the directory object for each version and the corresponding file system directory for each directory object. The versions that are new are created in this example. The versions that existed prior to this example were created in "Creating and Populating a Tablespace Repository" on page 16-2.

Table 16–2 Versions in the Tablespace Repository After inst2.net Reporting

Version	Directory Object	Corresponding File System Directory	New?
`v_q1fy2005`	`q1fy2005`	`/home/sales/q1fy2005`	No
`v_q1fy2005_r`	`q1fy2005_r`	`/home/sales/q1fy2005_r`	Yes
`v_q2fy2005`	`q2fy2005`	`/home/sales/q2fy2005`	No
`v_q2fy2005_r`	`q2fy2005_r`	`/home/sales/q2fy2005_r`	Yes

This example makes the following assumptions:

- The `inst1.net` and `inst2.net` databases exist.
- The `inst1.net` and `inst2.net` databases can access a shared file system.

- Networking is configured between the databases so that these databases can communicate with each other.
- A tablespace repository that contains a version of the sales tablespaces (`sales_tbs1` and `sales_tbs2`) for various quarters exists in the `inst1.net` database. This tablespace repository was created and populated in the example "Creating and Populating a Tablespace Repository" on page 16-2.

Complete the following steps:

1. Connect to `inst1.net`:

   ```
   CONNECT strmadmin/strmadminpw@inst1.net
   ```

   The administrative user must have the necessary privileges to create directory objects.

2. Create a directory object that will store the tablespace files for the first quarter in fiscal year 2005 on `inst1.net` after the `inst2.net` database has completed reporting on this quarter:

   ```
   CREATE OR REPLACE DIRECTORY q1fy2005_r AS '/home/sales/q1fy2005_r';
   ```

   The specified file system directory must exist when you create the directory objects.

3. Connect as an administrative user to the `inst2.net` database:

   ```
   CONNECT strmadmin/strmadminpw@inst2.net
   ```

   The administrative user must have the necessary privileges to run the procedures in the `DBMS_STREAMS_TABLESPACE_ADM` package, create directory objects, and create database links.

4. Create two directory objects for the first quarter in fiscal year 2005 on `inst2.net`. These directory objects must have the same names and correspond to the same directories on the shared file system as the directory objects used by the tablespace repository in the `inst1.net` database for the first quarter:

   ```
   CREATE OR REPLACE DIRECTORY q1fy2005 AS '/home/sales/q1fy2005';

   CREATE OR REPLACE DIRECTORY q1fy2005_r AS '/home/sales/q1fy2005_r';
   ```

5. Create a database link from `inst2.net` to the `inst1.net` database:

   ```
   CREATE DATABASE LINK inst1.net CONNECT TO strmadmin IDENTIFIED BY strmadminpw
      USING 'inst1.net';
   ```

6. Attach the tablespace set to the `inst2.net` database from the `strmadmin.sales` **file group** in the `inst1.net` database:

   ```
   DECLARE
     tbs_set DBMS_STREAMS_TABLESPACE_ADM.TABLESPACE_SET;
   BEGIN
     DBMS_STREAMS_TABLESPACE_ADM.ATTACH_TABLESPACES(
       file_group_name            => 'strmadmin.sales',
       version_name               => 'v_q1fy2005',
       datafiles_directory_object => 'q1fy2005_r',
       repository_db_link         => 'inst1.net',
       tablespace_names           => tbs_set);
   END;
   /
   ```

Notice that q1fy2005_r is specified for the datafiles_directory_object parameter. Therefore, the datafiles for the tablespaces and the export dump file are copied from the /home/sales/q1fy2005 location to the /home/sales/q1fy2005_r location by the procedure. The attached tablespaces in the inst2.net database use the datafiles in the /home/sales/q1fy2005_r location. The Data Pump import log file also is placed in this directory.

The attached tablespaces use the datafiles in the /home/sales/q1fy2005_r location. However, the v_q1fy2005 version of the tablespaces in the tablespace repository consists of the files in the original /home/sales/q1fy2005 location.

7. Make the tablespaces read/write at inst2.net:

```
ALTER TABLESPACE sales_tbs1 READ WRITE;

ALTER TABLESPACE sales_tbs2 READ WRITE;
```

8. Run the reports on the data in the sales tablespaces at the inst2.net database. The reports make changes to the tablespaces.

9. Detach the version of the tablespace set for the first quarter of 2005 from the inst2.net database:

```
DECLARE
  tbs_set  DBMS_STREAMS_TABLESPACE_ADM.TABLESPACE_SET;
BEGIN
  tbs_set(1) := 'sales_tbs1';
  tbs_set(2) := 'sales_tbs2';
  DBMS_STREAMS_TABLESPACE_ADM.DETACH_TABLESPACES(
    tablespace_names        => tbs_set,
    export_directory_object => 'q1fy2005_r',
    file_group_name         => 'strmadmin.sales',
    version_name            => 'v_q1fy2005_r',
    repository_db_link      => 'inst1.net');
END;
/
```

Only one version of a tablespace set can be attached to a database at a time. Therefore, the version of the sales tablespaces for the first quarter of 2005 must be detached from inst2.net before the version of this tablespace set for the second quarter of 2005 can be attached.

Also, notice that the specified export_directory_object is q1fy2005_r, and that the version_name is v_q1fy2005_r. After the detach operation, there are two versions of the tablespace files for the first quarter of 2005 stored in the tablespace repository on inst1.net: one version of the tablespace prior to reporting and one version after reporting. These two versions have different version names and are stored in different directory objects.

10. Connect to inst1.net, and create a directory object that will store the tablespace files for the second quarter in fiscal year 2005 on inst1.net after the inst2.net database has completed reporting on this quarter:

```
CONNECT strmadmin/strmadminpw@inst1.net

CREATE OR REPLACE DIRECTORY q2fy2005_r AS '/home/sales/q2fy2005_r';
```

The specified file system directory must exist when you create the directory object.

11. Connect to `inst2.net`, and create two directory objects for the second quarter in fiscal year 2005 at `inst2.net`. These directory objects must have the same names and correspond to the same directories on the shared file system as the directory objects used by the tablespace repository in the `inst1.net` database for the second quarter:

```
CONNECT strmadmin/strmadminpw@inst2.net

CREATE OR REPLACE DIRECTORY q2fy2005 AS '/home/sales/q2fy2005';

CREATE OR REPLACE DIRECTORY q2fy2005_r AS '/home/sales/q2fy2005_r';
```

12. Attach the tablespace set for the second quarter of 2005 to the `inst2.net` database from the `sales` file group in the `inst1.net` database:

```
DECLARE
  tbs_set DBMS_STREAMS_TABLESPACE_ADM.TABLESPACE_SET;
BEGIN
  DBMS_STREAMS_TABLESPACE_ADM.ATTACH_TABLESPACES(
    file_group_name            => 'strmadmin.sales',
    version_name               => 'v_q2fy2005',
    datafiles_directory_object => 'q2fy2005_r',
    repository_db_link         => 'inst1.net',
    tablespace_names           => tbs_set);
END;
/
```

13. Make the tablespaces read/write at `inst2.net`:

```
ALTER TABLESPACE sales_tbs1 READ WRITE;

ALTER TABLESPACE sales_tbs2 READ WRITE;
```

14. Run the reports on the data in the sales tablespaces at the `inst2.net` database. The reports make changes to the tablespace.

15. Detach the version of the tablespace set for the second quarter of 2005 from `inst2.net`:

```
DECLARE
  tbs_set  DBMS_STREAMS_TABLESPACE_ADM.TABLESPACE_SET;
BEGIN
  tbs_set(1) := 'sales_tbs1';
  tbs_set(2) := 'sales_tbs2';
  DBMS_STREAMS_TABLESPACE_ADM.DETACH_TABLESPACES(
    tablespace_names        => tbs_set,
    export_directory_object => 'q2fy2005_r',
    file_group_name         => 'strmadmin.sales',
    version_name            => 'v_q2fy2005_r',
    repository_db_link      => 'inst1.net');
END;
/
```

Steps 10-15 can be repeated whenever a quarter ends to run reports on each quarter.

Using a Tablespace Repository for Remote Reporting Without a Shared File System

This example runs reports at `inst3.net` on specific versions of the sales tablespaces stored in a **tablespace repository** at `inst1.net`. These two databases do not share a file system, and the reports that are run on `inst3.net` do not make any changes to the tablespace. Therefore, the tablespaces remain read-only at `inst3.net`, and, when the reports are complete, there is no need for a new **version** of the tablespace files in the tablespace repository on `inst1.net`.

Figure 16–4 provides an overview of how tablespaces in a tablespace repository are attached to a different database in this example:

Figure 16–4 Attaching Tablespaces Without a Shared File System

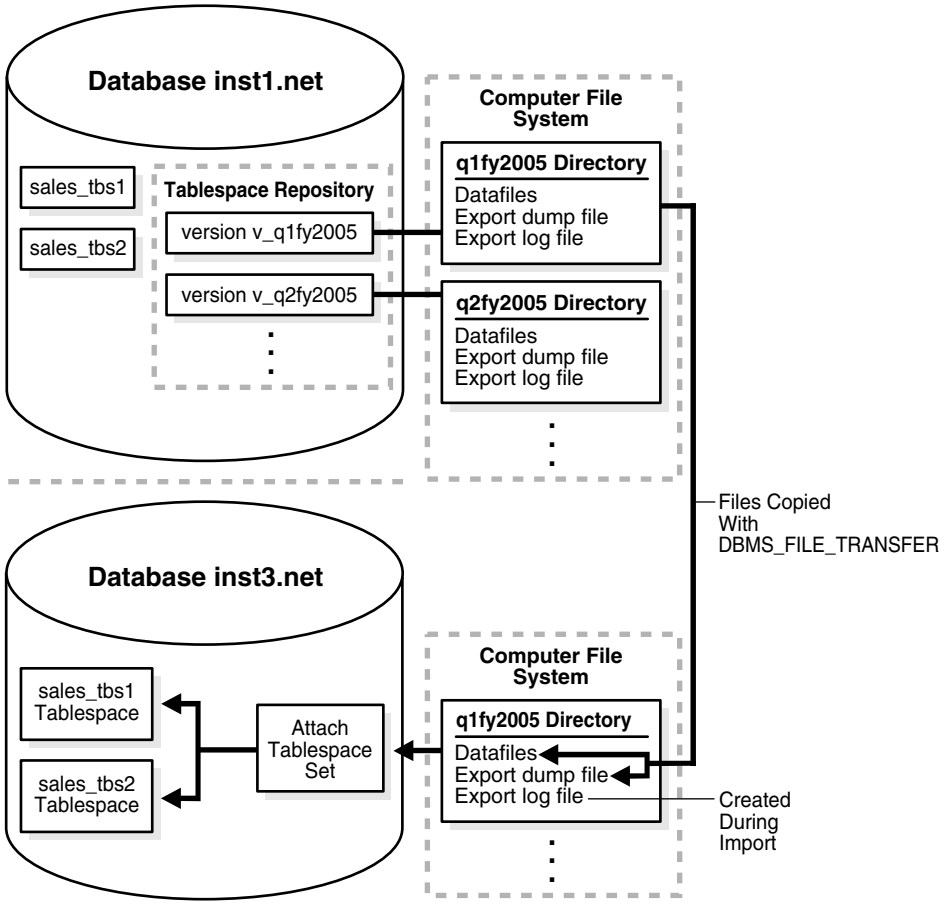


Table 16–3 shows the directory objects used in this example. It shows the existing directory objects that are associated with tablespace repository versions on the `inst1.net` database, and it shows the new directory objects created on the `inst3.net` database in this example. The directory objects that existed prior to this example were created in "Creating and Populating a Tablespace Repository" on page 16-2.

Table 16–3 Directory Objects Used in Example

Directory Object	Database	Version	Corresponding File System Directory	New?
q1fy2005	inst1.net	v_q1fy2005	/home/sales/q1fy2005	No
q2fy2005	inst1.net	v_q2fy2005	/home/sales/q2fy2005	No
q1fy2005	inst3.net	Not associated with a tablespace repository version	/usr/sales_data/fy2005q1	Yes
q2fy2005	inst3.net	Not associated with a tablespace repository version	/usr/sales_data/fy2005q2	Yes

This example makes the following assumptions:

- The inst1.net and inst3.net databases exist.
- The inst1.net and inst3.net databases do not share a file system.
- Networking is configured between the databases so that they can communicate with each other.
- The sales tablespaces (sales_tbs1 and sales_tbs2) exist in the inst1.net database.

Complete the following steps:

1. Connect as an administrative user to the inst3.net database:

   ```
   CONNECT strmadmin/strmadminpw@inst3.net
   ```

   The administrative user must have the necessary privileges to run the procedures in the DBMS_STREAMS_TABLESPACE_ADM package, create directory objects, and create database links.

2. Create a database link from inst3.net to the inst1.net database:

   ```
   CREATE DATABASE LINK inst1.net CONNECT TO strmadmin IDENTIFIED BY strmadminpw
      USING 'inst1.net';
   ```

3. Create a directory object for the first quarter in fiscal year 2005 on inst3.net. Although inst3.net is a remote database that does not share a file system with inst1.net, the directory object must have the same name as the directory object used by the tablespace repository in the inst1.net database for the first quarter. However, the directory paths of the directory objects on inst1.net and inst3.net do not need to match.

   ```
   CREATE OR REPLACE DIRECTORY q1fy2005 AS '/usr/sales_data/fy2005q1';
   ```

   The specified file system directory must exist when you create the directory object.

4. Connect as an administrative user to the inst1.net database:

   ```
   CONNECT strmadmin/strmadminpw@inst1.net
   ```

   The administrative user must have the necessary privileges to run the procedures in the DBMS_FILE_TRANSFER package and create database links. This example uses the DBMS_FILE_TRANSFER package to copy the tablespace files from inst1.net to inst3.net. If some other method is used to transfer the files, then the privileges to run the procedures in the DBMS_FILE_TRANSFER package are not required.

5. Create a database link from `inst1.net` to the `inst3.net` database:

```
CREATE DATABASE LINK inst3.net CONNECT TO strmadmin IDENTIFIED BY strmadminpw
   USING 'inst3.net';
```

This database link will be used to transfer files to the `inst3.net` database in Step 6.

6. Copy the datafile for each tablespace and the export dump file for the first quarter to the `inst3.net` database:

```
BEGIN
  DBMS_FILE_TRANSFER.PUT_FILE(
    source_directory_object      => 'q1fy2005',
    source_file_name             => 'sales_tbs1.dbf',
    destination_directory_object => 'q1fy2005',
    destination_file_name        => 'sales_tbs1.dbf',
    destination_database         => 'inst3.net');
  DBMS_FILE_TRANSFER.PUT_FILE(
    source_directory_object      => 'q1fy2005',
    source_file_name             => 'sales_tbs2.dbf',
    destination_directory_object => 'q1fy2005',
    destination_file_name        => 'sales_tbs2.dbf',
    destination_database         => 'inst3.net');
  DBMS_FILE_TRANSFER.PUT_FILE(
    source_directory_object      => 'q1fy2005',
    source_file_name             => 'expdat16.dmp',
    destination_directory_object => 'q1fy2005',
    destination_file_name        => 'expdat16.dmp',
    destination_database         => 'inst3.net');
END;
/
```

Before you run the `PUT_FILE` procedure for the export dump file, you can query the `DBA_FILE_GROUP_FILES` data dictionary view to determine the name and directory object of the export dump file. For example, run the following query to list this information for the export dump file in the `v_q1fy2005` version:

```
COLUMN FILE_NAME HEADING 'Export Dump|File Name' FORMAT A35
COLUMN FILE_DIRECTORY HEADING 'Directory Object' FORMAT A35

SELECT FILE_NAME, FILE_DIRECTORY FROM DBA_FILE_GROUP_FILES
  where FILE_GROUP_NAME = 'SALES' AND
        VERSION_NAME    = 'V_Q1FY2005';
```

7. Connect to `inst3.net` and attach the tablespace set for the first quarter of 2005 to the `inst3.net` database from the `sales` file group in the `inst1.net` database:

```
CONNECT strmadmin/strmadminpw@inst3.net

DECLARE
  tbs_set DBMS_STREAMS_TABLESPACE_ADM.TABLESPACE_SET;
BEGIN
  DBMS_STREAMS_TABLESPACE_ADM.ATTACH_TABLESPACES(
    file_group_name            => 'strmadmin.sales',
    version_name               => 'v_q1fy2005',
    datafiles_directory_object => 'q1fy2005',
    repository_db_link         => 'inst1.net',
    tablespace_names           => tbs_set);
END;
/
```

The tablespaces are read-only when they are attached. Because the reports on `inst3.net` do not change the tablespaces, the tablespaces can remain read-only.

8. Run the reports on the data in the sales tablespaces at the `inst3.net` database.
9. Drop the tablespaces and their contents at `inst3.net`:

```
DROP TABLESPACE sales_tbs1 INCLUDING CONTENTS;

DROP TABLESPACE sales_tbs2 INCLUDING CONTENTS;
```

The tablespaces are dropped from the `inst3.net` database, but the tablespace files remain in the directory object.

10. Create a directory object for the second quarter in fiscal year 2005 on `inst3.net`. The directory object must have the same name as the directory object used by the tablespace repository in the `inst1.net` database for the second quarter. However, the directory paths of the directory objects on `inst1.net` and `inst3.net` do not need to match.

```
CREATE OR REPLACE DIRECTORY q2fy2005 AS '/usr/sales_data/fy2005q2';
```

The specified file system directory must exist when you create the directory object.

11. Connect to the `inst1.net` database and copy the datafile and the export dump file for the second quarter to the `inst3.net` database:

```
CONNECT strmadmin/strmadminpw@inst1.net

BEGIN
  DBMS_FILE_TRANSFER.PUT_FILE(
    source_directory_object      => 'q2fy2005',
    source_file_name             => 'sales_tbs1.dbf',
    destination_directory_object => 'q2fy2005',
    destination_file_name        => 'sales_tbs1.dbf',
    destination_database         => 'inst3.net');
  DBMS_FILE_TRANSFER.PUT_FILE(
    source_directory_object      => 'q2fy2005',
    source_file_name             => 'sales_tbs2.dbf',
    destination_directory_object => 'q2fy2005',
    destination_file_name        => 'sales_tbs2.dbf',
    destination_database         => 'inst3.net');
  DBMS_FILE_TRANSFER.PUT_FILE(
    source_directory_object      => 'q2fy2005',
    source_file_name             => 'expdat18.dmp',
    destination_directory_object => 'q2fy2005',
    destination_file_name        => 'expdat18.dmp',
    destination_database         => 'inst3.net');
END;
/
```

Before you run the `PUT_FILE` procedure for the export dump file, you can query the `DBA_FILE_GROUP_FILES` data dictionary view to determine the name and directory object of the export dump file. For example, run the following query to list this information for the export dump file in the `v_q2fy2005` version:

```
COLUMN FILE_NAME HEADING 'Export Dump|File Name' FORMAT A35
COLUMN FILE_DIRECTORY HEADING 'Directory Object' FORMAT A35

SELECT FILE_NAME, FILE_DIRECTORY FROM DBA_FILE_GROUP_FILES
  where FILE_GROUP_NAME = 'SALES' AND
        VERSION_NAME    = 'V_Q2FY2005';
```

12. Attach the tablespace set for the second quarter of 2005 to the `inst3.net` database from the `sales` **file group** in the `inst1.net` database:

```
CONNECT strmadmin/strmadminpw@inst3.net

DECLARE
  tbs_set DBMS_STREAMS_TABLESPACE_ADM.TABLESPACE_SET;
BEGIN
  DBMS_STREAMS_TABLESPACE_ADM.ATTACH_TABLESPACES(
    file_group_name            => 'strmadmin.sales',
    version_name               => 'v_q2fy2005',
    datafiles_directory_object => 'q2fy2005',
    repository_db_link         => 'inst1.net',
    tablespace_names           => tbs_set);
END;
/
```

    The tablespaces are read-only when they are attached. Because the reports on `inst3.net` do not change the tablespace, the tablespaces can remain read-only.

13. Run the reports on the data in the sales tablespaces at the `inst3.net` database.

14. Drop the tablespaces and their contents:

```
DROP TABLESPACE sales_tbs1 INCLUDING CONTENTS;

DROP TABLESPACE sales_tbs2 INCLUDING CONTENTS;
```

    The tablespaces are dropped from the `inst3.net` database, but the tablespace files remain in the directory object.

Steps 10-14 can be repeated whenever a quarter ends to run reports on each quarter.

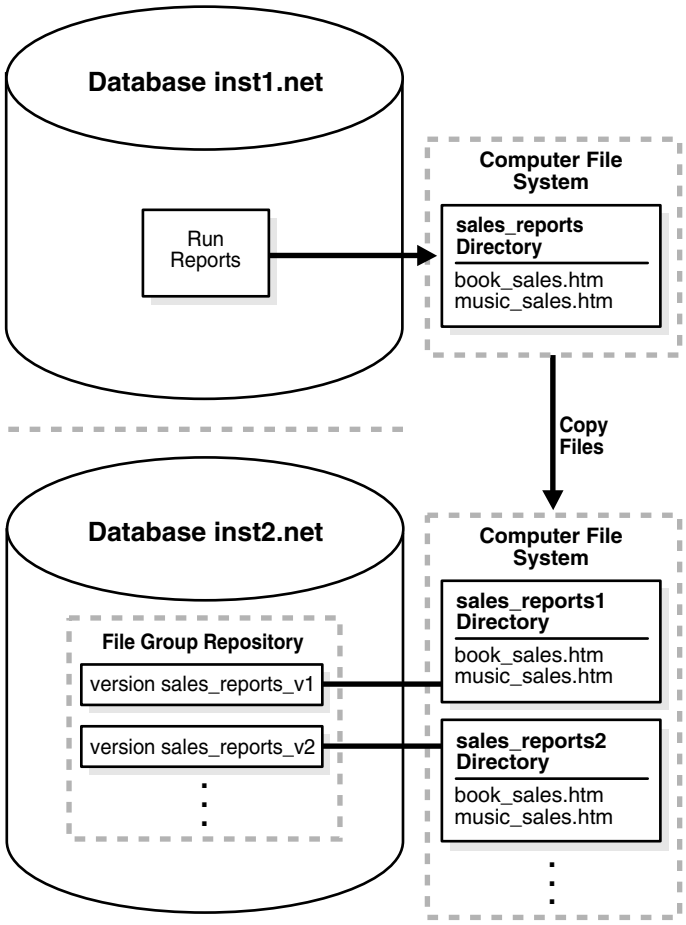
Using a File Group Repository

The `DBMS_FILE_GROUP` package can create a **file group repository**, add versioned **file group**s to the repository, and copy versioned file groups from the repository. This section illustrates how to use a file group repository with a scenario that stores reports in the repository.

In this scenario, a business sells books and music over the internet. The business runs weekly reports on the sales data in the `inst1.net` database and stores these reports in two HTML files on a computer file system. The `book_sales.htm` file contains the report for book sales, and the `music_sales.htm` file contains the report for music sales. The business wants to store these weekly reports in a file group repository at the `inst2.net` remote database. Every week, the two reports are generated on the `inst1.net` database, transferred to the computer system running the `inst2.net` database, and added to the repository as a file group **version**. The file group repository stores all of the file group versions that contain the reports for each week.

Figure 16–5 provides an overview of the file group repository created in this example:

Figure 16–5 Example File Group Repository



The benefits of the file group repository are that it stores metadata about each file group version in the data dictionary and provides a standard interface for managing the file group versions. For example, when the business needs to view a specific sales report, it can query the data dictionary in the `inst2.net` database to determine the location of the report on the computer file system.

Table 16–4 shows the directory objects created in this example. It shows the directory object created on the `inst1.net` database to store new reports, and it shows the directory objects that are associated with file group repository versions on the `inst2.net` database.

Table 16–4 Directory Objects Created in Example

Directory Object	Database	Version	Corresponding File System Directory
`sales_reports`	`inst1.net`	Not associated with a file group repository version	`/home/sales_reports`
`sales_reports1`	`inst2.net`	`sales_reports_v1`	`/home/sales_reports/fg1`
`sales_reports2`	`inst2.net`	`sales_reports_v1`	`/home/sales_reports/fg2`

This example makes the following assumptions:

- The `inst1.net` and `inst2.net` databases exist.
- The `inst1.net` and `inst2.net` databases do not share a file system.

- Networking is configured between the databases so that they can communicate with each other.
- The `inst1.net` database runs reports on the books and music sales data in the database and stores the reports as HTML files on the computer file system.

The following steps configure and populate a file group repository at a remote database:

1. Connect as an administrative user to the remote database that will contain the file group repository:

   ```
   CONNECT strmadmin/strmadminpw@inst2.net
   ```

   The administrative user must have the necessary privileges to create directory objects and run the procedures in the `DBMS_FILE_GROUP` package.

2. Create a directory object to hold the first version of the file group:

   ```
   CREATE OR REPLACE DIRECTORY sales_reports1 AS '/home/sales_reports/fg1';
   ```

   The specified file system directory must exist when you create the directory object.

3. Connect as an administrative user to the database that runs the reports:

   ```
   CONNECT strmadmin/strmadminpw@inst1.net
   ```

   The administrative user must have the necessary privileges to create directory objects.

4. Create a directory object to hold the latest reports:

   ```
   CREATE OR REPLACE DIRECTORY sales_reports AS '/home/sales_reports';
   ```

   The specified file system directory must exist when you create the directory object.

5. Create a database link to the `inst2.net` database:

   ```
   CREATE DATABASE LINK inst2.net CONNECT TO strmadmin IDENTIFIED BY strmadminpw
      USING 'inst2.net';
   ```

6. Run the reports on the `inst1.net` database. Running the reports should place the `book_sales.htm` and `music_sales.htm` files in the directory specified in Step 4.

7. Transfer the report files from the computer system running the `inst1.net` database to the computer system running the `inst2.net` database using file transfer protocol (FTP) or some other method. Make sure the files are copied to the directory that corresponds to the directory object created in Step 2.

8. Connect in SQL*Plus to `inst2.net`:

   ```
   CONNECT strmadmin/strmadminpw@inst2.net
   ```

9. Create the file group repository that will contain the reports:

   ```
   BEGIN
     DBMS_FILE_GROUP.CREATE_FILE_GROUP(
       file_group_name => 'strmadmin.reports');
   END;
   /
   ```

The `reports` file group repository is created with the following default properties:

- The minimum number of versions in the repository is 2. When the file group is purged, the number of versions cannot drop below 2.
- The maximum number of versions is infinite. A file group version is not purged because of the number of versions in the of the file group in the repository.
- The retention days is infinite. A file group version is not purged because of the amount of time it has been in the repository.

10. Create the first version of the file group:

```
BEGIN
  DBMS_FILE_GROUP.CREATE_VERSION(
    file_group_name => 'strmadmin.reports',
    version_name    => 'sales_reports_v1',
    comments        => 'Sales reports for week of 06-FEB-2005');
END;
/
```

11. Add the report files to the file group version:

```
BEGIN
 DBMS_FILE_GROUP.ADD_FILE(
    file_group_name  => 'strmadmin.reports',
    file_name        => 'book_sales.htm',
    file_type        => 'HTML',
    file_directory   => 'sales_reports1',
    version_name     => 'sales_reports_v1');
 DBMS_FILE_GROUP.ADD_FILE(
    file_group_name  => 'strmadmin.reports',
    file_name        => 'music_sales.htm',
    file_type        => 'HTML',
    file_directory   => 'sales_reports1',
    version_name     => 'sales_reports_v1');
END;
/
```

12. Create a directory object on `inst2.net` to hold the next version of the file group:

```
CREATE OR REPLACE DIRECTORY sales_reports2 AS '/home/sales_reports/fg2';
```

The specified file system directory must exist when you create the directory object.

13. At the end of the next week, run the reports on the `inst1.net` database. Running the reports should place new `book_sales.htm` and `music_sales.htm` files in the directory specified in Step 4. If necessary, remove the old files from this directory before running the reports.

14. Transfer the report files from the computer system running the `inst1.net` database to the computer system running the `inst2.net` database using file transfer protocol (FTP) or some other method. Make sure the files are copied to the directory that corresponds to the directory object created in Step 12.

15. While connected in SQL*Plus to `inst2.net` as an administrative user, create the next version of the file group:

```
BEGIN
  DBMS_FILE_GROUP.CREATE_VERSION(
    file_group_name => 'strmadmin.reports',
    version_name    => 'sales_reports_v2',
    comments        => 'Sales reports for week of 13-FEB-2005');
END;
/
```

16. Add the report files to the file group version:

```
BEGIN
 DBMS_FILE_GROUP.ADD_FILE(
    file_group_name  => 'strmadmin.reports',
    file_name        => 'book_sales.htm',
    file_type        => 'HTML',
    file_directory   => 'sales_reports2',
    version_name     => 'sales_reports_v2');
 DBMS_FILE_GROUP.ADD_FILE(
    file_group_name  => 'strmadmin.reports',
    file_name        => 'music_sales.htm',
    file_type        => 'HTML',
    file_directory   => 'sales_reports2',
    version_name     => 'sales_reports_v2');
END;
/
```

The file group repository now contains two versions of the file group that contains the sales report files. Repeat steps 12-16 to add new versions of the file group to the repository.

See Also:

- "File Group Repository" on page 8-4
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_FILE_GROUP` package

17

Other Streams Management Tasks

This chapter provides instructions for performing full database export/import in a Streams environment. This chapter also provides instructions for removing a Streams configuration.

This chapter contains these topics:

- Performing Full Database Export/Import in a Streams Environment
- Removing a Streams Configuration

Each task described in this chapter should be completed by a Streams administrator that has been granted the appropriate privileges, unless specified otherwise.

See Also: "Configuring a Streams Administrator" on page 10-1

Performing Full Database Export/Import in a Streams Environment

This section describes how to perform a full database export/import on a database that is running one or more Streams **capture process**es, **propagation**s, or **apply process**es. These instructions pertain to a full database export/import where the import database and export database are running on different computers, and the import database replaces the export database. The global name of the import database and the global name of the export database must match. These instructions assume that both databases already exist. The export/import described in this section can be performed using Data Pump Export/Import utilities or the original Export/Import utilities.

Note: If you want to add a database to an existing Streams environment, then do not use the instructions in this section. Instead, see *Oracle Streams Replication Administrator's Guide*.

See Also:

- *Oracle Streams Replication Administrator's Guide* for more information about export/import parameters that are relevant to Streams
- *Oracle Database Utilities* for more information about performing a full database export/import

Complete the following steps to perform a full database export/import on a database that is using Streams:

1. If the export database contains any **destination queue**s for **propagation**s from other databases, then stop each propagation that propagates **message**s to the export database. You can stop a propagation using the `STOP_PROPAGATION` procedure in the `DBMS_PROPAGATION_ADM` package.
2. Make the necessary changes to your network configuration so that the database links used by the **propagation job**s you disabled in Step 1 point to the computer running the import database.

   To complete this step, you might need to re-create the database links used by these propagation jobs or modify your Oracle networking files at the databases that contain the **source queue**s.
3. Notify all users to stop making data manipulation language (DML) and data definition language (DDL) changes to the export database, and wait until these changes have stopped.
4. Make a note of the current export database system change number (SCN). You can determine the current SCN using the `GET_SYSTEM_CHANGE_NUMBER` function in the `DBMS_FLASHBACK` package. For example:

   ```
   SET SERVEROUTPUT ON SIZE 1000000
   DECLARE
     current_scn NUMBER;
   BEGIN
     current_scn:= DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER;
         DBMS_OUTPUT.PUT_LINE('Current SCN: ' || current_scn);
   END;
   /
   ```

   In this example, assume that current SCN returned is `7000000`.

   After completing this step, do not stop any **capture process** running on the export database. Step 7c instructs you to use the `V$STREAMS_CAPTURE` dynamic performance view to ensure that no DML or DDL changes were made to the database after Step 3. The information about a capture process in this view is reset if the capture process is stopped and restarted.

   For the check in Step 7c to be valid, this information should not be reset for any capture process. To prevent a capture process from stopping automatically, you might need to set the `message_limit` and `time_limit` capture process parameters to `infinite` if these parameters are set to another value for any capture process.
5. If any **downstream capture process**es are capturing changes that originated at the export database, then make sure the log file containing the SCN determined in Step 4 has been transferred to the **downstream database** and added to the capture process session. See "Displaying the Registered Redo Log Files for Each Capture Process" on page 20-7 for queries that can determine this information.
6. If the export database is not running any apply processes, and is not propagating **user-enqueued message**s, then start the full database export now. Make sure that the `FULL` export parameter is set to `y` so that the required Streams metadata is exported.

   If the export database is running one or more apply processes or is propagating user-enqueued messages, then do not start the export and proceed to the next step.

7. If the export database is the **source database** for changes captured by any capture processes, then complete the following steps for each capture process:

   a. Wait until the capture process has scanned past the redo record that corresponds to the SCN determined in Step 4. You can view the SCN of the redo record last scanned by a capture process by querying the CAPTURE_MESSAGE_NUMBER column in the V$STREAMS_CAPTURE dynamic performance view. Make sure the value of CAPTURE_MESSAGE_NUMBER is greater than or equal to the SCN determined in Step 4 before you continue.

   b. Monitor the Streams environment until the apply process at the **destination database** has applied all of the changes from the capture database. For example, if the name of the capture process is capture, the name of the apply process is apply, the global name of the destination database is dest.net, and the SCN value returned in Step 4 is 7000000, then run the following query at the capture database:

```
CONNECT strmadmin/strmadminpw

SELECT cap.ENQUEUE_MESSAGE_NUMBER
  FROM V$STREAMS_CAPTURE cap
  WHERE cap.CAPTURE_NAME = 'CAPTURE' AND
        cap.ENQUEUE_MESSAGE_NUMBER IN (
          SELECT DEQUEUED_MESSAGE_NUMBER
          FROM V$STREAMS_APPLY_READER@dest.net reader,
               V$STREAMS_APPLY_COORDINATOR@dest.net coord
          WHERE reader.APPLY_NAME = 'APPLY' AND
            reader.DEQUEUED_MESSAGE_NUMBER = reader.OLDEST_SCN_NUM AND
            coord.APPLY_NAME = 'APPLY' AND
            coord.LWM_MESSAGE_NUMBER = coord.HWM_MESSAGE_NUMBER AND
            coord.APPLY# = reader.APPLY#) AND
          cap.CAPTURE_MESSAGE_NUMBER >= 7000000;
```

   When this query returns a row, all of the changes from the capture database have been applied at the destination database, and you can move on to the next step.

   If this query returns no results for an inordinately long time, then make sure the **Streams client**s in the environment are enabled by querying the STATUS column in the DBA_CAPTURE view at the source database and the DBA_APPLY view at the destination database. You can check the status of the propagation by running the query in "Displaying the Schedule for a Propagation Job" on page 21-16.

   If a Streams client is disabled, then try restarting it. If a Streams client will not restart, then troubleshoot the environment using the information in Chapter 18, "Troubleshooting a Streams Environment".

   The query in this step assumes that a database link accessible to the Streams administrator exists between the capture database and the destination database. If such a database link does not exist, then you can perform two separate queries at the capture database and destination database.

   c. Verify that the enqueue message number of each capture process is less than or equal to the SCN determined in Step 4. You can view the enqueue message number for each capture process by querying the ENQUEUE_MESSAGE_NUMBER column in the V$STREAMS_CAPTURE dynamic performance view.

   If the enqueue message number of each capture process is less than or equal to the SCN determined in Step 4, then proceed to Step 9.

However, if the enqueue message number of any capture process is higher than the SCN determined in Step 4, then one or more DML or DDL changes were made after the SCN determined in Step 4, and these changes were captured and enqueued by a capture process. In this case, perform all of the steps in this section again, starting with Step 1 on page 17-2.

Note: For this verification to be valid, each capture process must have been running uninterrupted since Step 4.

---

8. If any downstream capture processes captured changes that originated at the export database, then drop these downstream capture processes. You will re-create them in Step 14a.
9. If the export database has any propagations that are propagating user-enqueued messages, then stop these propagations using the `STOP_PROPAGATION` procedure in the `DBMS_PROPAGATION` package.
10. If the export database is running one or more apply processes, or is propagating user-enqueued messages, then start the full database export now. Make sure that the `FULL` export parameter is set to `y` so that the required Streams metadata is exported. If you already started the export in Step 6, then proceed to Step 11.
11. When the export is complete, transfer the export dump file to the computer running the import database.
12. Perform the full database import. Make sure that the `STREAMS_CONFIGURATION` and `FULL` import parameters are both set to `y` so that the required Streams metadata is imported. The default setting is `y` for the `STREAMS_CONFIGURATION` import parameter. Also, make sure no DML or DDL changes are made to the import database during the import.
13. If any downstream capture processes are capturing changes that originated at the database, then make the necessary changes so that log files are transferred from the import database to the downstream database. See "Preparing to Copy Redo Log Files for Archived-Log Downstream Capture" on page 11-13 for instructions.
14. Re-create downstream capture processes:
    a. Re-create any downstream capture processes that you dropped in Step 8, if necessary. These dropped downstream capture processes were capturing changes that originated at the export database. Configure the re-created downstream capture processes to capture changes that originate at the import database.
    b. Re-create in the import database any downstream capture processes that were running in the export database, if necessary. If the export database had any downstream capture processes, then those downstream capture processes were not exported.

    See Also: "Creating a Capture Process" on page 11-2 for information about creating a downstream capture process
15. If any local or downstream capture processes will capture changes that originate at the database, then, at the import database, prepare the database objects whose changes will be captured for **instantiation**. See *Oracle Streams Replication Administrator's Guide* for information about preparing database objects for instantiation.

16. Let users access the import database, and shut down the export database.
17. Enable any propagation jobs you disabled in Steps 1 and 9.
18. If you reset the value of a `message_limit` or `time_limit` capture process parameter in Step 4, then, at the import database, reset these parameters to their original settings.

Removing a Streams Configuration

You run the `REMOVE_STREAMS_CONFIGURATION` procedure in the `DBMS_STREAMS_ADM` package to remove a Streams configuration at the local database.

Attention: Running this procedure is dangerous. You should run this procedure only if you are sure you want to remove the entire Streams configuration at a database.

To remove the Streams configuration at the local database, run the following procedure:

```
EXEC DBMS_STREAMS_ADM.REMOVE_STREAMS_CONFIGURATION();
```

See Also: *Oracle Database PL/SQL Packages and Types Reference* for detailed information about the actions performed by the `REMOVE_STREAMS_CONFIGURATION` procedure

18

Troubleshooting a Streams Environment

This chapter contains information about identifying and resolving common problems in a Streams environment.

This chapter contains these topics:

- Troubleshooting Capture Problems
- Troubleshooting Propagation Problems
- Troubleshooting Apply Problems
- Troubleshooting Problems with Rules and Rule-Based Transformations
- Checking the Trace Files and Alert Log for Problems

> **See Also:** *Oracle Streams Replication Administrator's Guide* for more information about troubleshooting Streams **replication** environments

Troubleshooting Capture Problems

If a **capture process** is not capturing changes as expected, or if you are having other problems with a capture process, then use the following checklist to identify and resolve capture problems:

- Is the Capture Process Enabled?
- Is the Capture Process Current?
- Are Required Redo Log Files Missing?
- Is a Downstream Capture Process Waiting for Redo Data?
- Are You Trying to Configure Downstream Capture without DBMS_CAPTURE_ADM?
- Are More Actions Required for Downstream Capture without a Database Link?

> **See Also:**
>
> - Chapter 2, "Streams Capture Process"
> - Chapter 11, "Managing a Capture Process"
> - Chapter 20, "Monitoring Streams Capture Processes"

Is the Capture Process Enabled?

A **capture process** captures changes only when it is enabled.

You can check whether a capture process is enabled, disabled, or aborted by querying the DBA_CAPTURE data dictionary view. For example, to check whether a capture process named capture is enabled, run the following query:

```
SELECT STATUS FROM DBA_CAPTURE WHERE CAPTURE_NAME = 'CAPTURE';
```

If the capture process is disabled, then your output looks similar to the following:

```
STATUS
--------
DISABLED
```

If the capture process is disabled, then try restarting it. If the capture process is aborted, then you might need to correct an error before you can restart it successfully.

To determine why the capture process aborted, query the DBA_CAPTURE data dictionary view or check the trace file for the capture process. The following query shows when the capture process aborted and the error that caused it to abort:

```
COLUMN CAPTURE_NAME HEADING 'Capture|Process|Name' FORMAT A10
COLUMN STATUS_CHANGE_TIME HEADING 'Abort Time'
COLUMN ERROR_NUMBER HEADING 'Error Number' FORMAT 99999999
COLUMN ERROR_MESSAGE HEADING 'Error Message' FORMAT A40

SELECT CAPTURE_NAME, STATUS_CHANGE_TIME, ERROR_NUMBER, ERROR_MESSAGE
  FROM DBA_CAPTURE WHERE STATUS='ABORTED';
```

See Also:

- "Starting a Capture Process" on page 11-24
- "Checking the Trace Files and Alert Log for Problems" on page 18-21
- "Streams Capture Processes and Oracle Real Application Clusters" on page 2-21 for information about restarting a capture process in an Oracle Real Application Clusters environment

Is the Capture Process Current?

If a **capture process** has not captured recent changes, then the cause might be that the capture process has fallen behind. To check, you can query the V$STREAMS_CAPTURE dynamic performance view. If capture process latency is high, then you might be able to improve performance by adjusting the setting of the parallelism capture process parameter.

See Also:

- "Determining Redo Log Scanning Latency for Each Capture Process" on page 20-13
- "Determining Message Enqueuing Latency for Each Capture Process" on page 20-14
- "Capture Process Parallelism" on page 2-40
- "Setting a Capture Process Parameter" on page 11-28

Are Required Redo Log Files Missing?

When a **capture process** is started or restarted, it might need to scan redo log files that were generated before the log file that contains the **start SCN**. You can query the DBA_CAPTURE data dictionary view to determine the **first SCN** and start SCN for a capture process. Removing required redo log files before they are scanned by a capture process causes the capture process to abort and results in the following error in a capture process trace file:

```
ORA-01291: missing logfile
```

If you see this error, then try restoring any missing redo log file and restarting the capture process. You can check the V$LOGMNR_LOGS dynamic performance view to determine the missing SCN range, and add the relevant redo log files. A capture process needs the redo log file that includes the **required checkpoint SCN** and all subsequent redo log files. You can query the REQUIRED_CHECKPOINT_SCN column in the DBA_CAPTURE data dictionary view to determine the required checkpoint SCN for a capture process.

If you are using the flash recovery area feature of Recovery Manager (RMAN) on a **source database** in a Streams environment, then RMAN might delete archived redo log files that are required by a capture process. RMAN might delete these files when the disk space used by the recovery-related files is nearing the specified disk quota for the flash recovery area. To prevent this problem in the future, complete one or more of the following actions:

- Increase the disk quota for the flash recovery area. Increasing the disk quota makes it less likely that RMAN will delete a required archived redo log file, but it will not always prevent the problem.
- Configure the source database to store archived redo log files in a location other than the flash recovery area. A **local capture process** will be able to use the log files in the other location if the required log files are missing in the flash recovery area. In this case, a database administrator must manage the log files manually in the other location.

See Also:

- "ARCHIVELOG Mode and a Capture Process" on page 2-38
- "First SCN and Start SCN" on page 2-19
- "Displaying the Registered Redo Log Files for Each Capture Process" on page 20-7
- *Oracle Database Backup and Recovery Basics* and *Oracle Database Backup and Recovery Advanced User's Guide* for more information about the flash recovery area feature

Is a Downstream Capture Process Waiting for Redo Data?

If a **downstream capture process** is not capturing changes, then it might be waiting for redo data to scan. Redo log files can be registered implicitly or explicitly for a downstream capture process. Redo log files registered implicitly typically are registered in one of the following ways:

- For a **real-time downstream capture process**, redo transport services use the log writer process (LGWR) to transfer the redo data from the **source database** to the standby redo log at the **downstream database**. Next, the archiver at the downstream database registers the redo log files with the downstream capture process when it archives them.
- For an **archived-log downstream capture process**, redo transport services transfer the archived redo log files from the source database to the downstream database and register the archived redo log files with the downstream capture process.

If redo log files are registered explicitly for a downstream capture process, then you must manually transfer the redo log files to the downstream database and register them with the downstream capture process.

Regardless of whether the redo log files are registered implicitly or explicitly, the downstream capture process can capture changes made to the source database only if the appropriate redo log files are registered with the downstream capture process. You can query the `V$STREAMS_CAPTURE` dynamic performance view to determine whether a downstream capture process is waiting for a redo log file. For example, run the following query for a downstream capture process named `strm05_capture`:

```
SELECT STATE FROM V$STREAMS_CAPTURE WHERE CAPTURE_NAME='STRM05_CAPTURE';
```

If the capture process state is either `WAITING FOR DICTIONARY REDO` or `WAITING FOR REDO`, then verify that the redo log files have been registered with the downstream capture process by querying the `DBA_REGISTERED_ARCHIVED_LOG` and `DBA_CAPTURE` data dictionary views. For example, the following query lists the redo log files currently registered with the `strm05_capture` downstream capture process:

```
COLUMN SOURCE_DATABASE HEADING 'Source|Database' FORMAT A15
COLUMN SEQUENCE# HEADING 'Sequence|Number' FORMAT 9999999
COLUMN NAME HEADING 'Archived Redo Log|File Name' FORMAT A30
COLUMN DICTIONARY_BEGIN HEADING 'Dictionary|Build|Begin' FORMAT A10
COLUMN DICTIONARY_END HEADING 'Dictionary|Build|End' FORMAT A10

SELECT r.SOURCE_DATABASE,
       r.SEQUENCE#,
       r.NAME,
       r.DICTIONARY_BEGIN,
       r.DICTIONARY_END
  FROM DBA_REGISTERED_ARCHIVED_LOG r, DBA_CAPTURE c
  WHERE c.CAPTURE_NAME = 'STRM05_CAPTURE' AND
        r.CONSUMER_NAME = c.CAPTURE_NAME;
```

If this query does not return any rows, then no redo log files are registered with the capture process currently. If you configured redo transport services to transfer redo data from the source database to the downstream database for this capture process, then make sure the redo transport services are configured correctly. If the redo transport services are configured correctly, then run the `ALTER SYSTEM ARCHIVE LOG CURRENT` statement at the source database to archive a log file. If you did not configure redo transport services to transfer redo data, then make sure the method you are using for log file transfer and registration is working properly. You can

register log files explicitly using an ALTER DATABASE REGISTER LOGICAL LOGFILE statement.

If the downstream capture process is waiting for redo, then it also is possible that there is a problem with the network connection between the source database and the downstream database. There also might be a problem with the log file transfer method. Check your network connection and log file transfer method to ensure that they are working properly.

If you configured a real-time downstream capture process, and no redo log files are registered with the capture process, then try switching the log file at the source database. You might need to switch the log file more than once if there is little or no activity at the source database.

Also, if you plan to use a downstream capture process to capture changes to historical data, then consider the following additional issues:

- Both the source database that generates the redo log files and the database that runs a downstream capture process must be Oracle Database 10*g* databases.
- The start of a data dictionary build must be present in the oldest redo log file added, and the capture process must be configured with a **first SCN** that matches the start of the data dictionary build.
- The database objects for which the capture process will capture changes must be prepared for **instantiation** at the source database, not at the downstream database. In addition, you cannot specify a time in the past when you prepare objects for instantiation. Objects are always prepared for instantiation at the current database SCN, and only changes to a database object that occurred after the object was prepared for instantiation can be captured by a capture process.

See Also:

- "Local Capture and Downstream Capture" on page 2-12
- Capture Process States on page 2-24
- "Creating an Archived-Log Downstream Capture Process that Assigns Logs Implicitly" on page 11-13
- "Creating an Archived-Log Downstream Capture Process that Assigns Logs Explicitly" on page 11-18

Are You Trying to Configure Downstream Capture without DBMS_CAPTURE_ADM?

You must use the CREATE_CAPTURE procedure in the DBMS_CAPTURE_ADM package to create a **downstream capture process**. If you try to create a capture process using a procedure in the DBMS_STREAMS_ADM package and specify a **source database** name that does not match the global name of the local database, then Oracle returns the following error:

```
ORA-26678: Streams capture process must be created first
```

To correct the problem, use the CREATE_CAPTURE procedure in the DBMS_CAPTURE_ADM package to create the downstream capture process.

If you are trying to create a **local capture process** using a procedure in the DBMS_STREAMS_ADM package, and you encounter this error, then make sure the database name specified in the source_database parameter of the procedure you are running matches the global name of the local database.

See Also: "Creating a Capture Process" on page 11-2

Are More Actions Required for Downstream Capture without a Database Link?

When downstream capture is configured with a database link, the database link can be used to perform operations at the **source database** and obtain information from the source database automatically. When downstream capture is configured without a database link, these actions must be performed manually, and the information must be obtained manually. If you do not complete these actions manually, then errors result when you try to create the downstream **capture process**.

Specifically, the following actions must be performed manually when you configure downstream capture without a database link:

- In certain situations, you must run the `DBMS_CAPTURE_ADM.BUILD` procedure at the source database to extract the data dictionary at the source database to the redo log before a capture process is created.
- You must prepare the source database objects for **instantiation**.
- You must obtain the **first SCN** for the **downstream capture process** and specify the first SCN using the `first_scn` parameter when you create the capture process with the `CREATE_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package.

See Also: "Creating an Archived-Log Downstream Capture Process" on page 11-15

Troubleshooting Propagation Problems

If a **propagation** is not propagating changes as expected, then use the following checklist to identify and resolve propagation problems:

- Does the Propagation Use the Correct Source and Destination Queue?
- Is the Propagation Enabled?
- Are There Enough Job Queue Processes?
- Is Security Configured Properly for the ANYDATA Queue?

See Also:

- Chapter 3, "Streams Staging and Propagation"
- Chapter 12, "Managing Staging and Propagation"
- "Monitoring Streams Propagations and Propagation Jobs" on page 21-13

Does the Propagation Use the Correct Source and Destination Queue?

If messages are not appearing in the **destination queue** for a **propagation** as expected, then the propagation might not be configured to propagate messages from the correct **source queue** to the correct destination queue.

For example, to check the source queue and destination queue for a propagation named `dbs1_to_dbs2`, run the following query:

```
COLUMN SOURCE_QUEUE HEADING 'Source Queue' FORMAT A35
COLUMN DESTINATION_QUEUE HEADING 'Destination Queue' FORMAT A35
```

```
SELECT
  p.SOURCE_QUEUE_OWNER||'.'||
    p.SOURCE_QUEUE_NAME||'@'||
    g.GLOBAL_NAME SOURCE_QUEUE,
  p.DESTINATION_QUEUE_OWNER||'.'||
    p.DESTINATION_QUEUE_NAME||'@'||
    p.DESTINATION_DBLINK DESTINATION_QUEUE
  FROM DBA_PROPAGATION p, GLOBAL_NAME g
  WHERE p.PROPAGATION_NAME = 'DBS1_TO_DBS2';
```

Your output looks similar to the following:

```
Source Queue                          Destination Queue
------------------------------------ ------------------------------------
STRMADMIN.STREAMS_QUEUE@DBS1.NET     STRMADMIN.STREAMS_QUEUE@DBS2.NET
```

If the propagation is not using the correct queues, then create a new propagation. You might need to remove the existing propagation if it is not appropriate for your environment.

> **See Also:** "Creating a Propagation Between Two ANYDATA Queues" on page 12-7

Is the Propagation Enabled?

For a **propagation job** to propagate messages, the propagation must be enabled. If messages are not being propagated by a **propagation** as expected, then the propagation might not be enabled.

You can find the following information about a propagation:

- The database link used to propagate messages from the **source queue** to the **destination queue**
- Whether the propagation is `ENABLED`, `DISABLED`, or `ABORTED`
- The date of the last error, if there are any propagation errors
- The error message of the last error, if there are any propagation errors

For example, to check whether a propagation named `streams_propagation` is enabled, run the following query:

```
COLUMN DESTINATION_DBLINK HEADING 'Database|Link'      FORMAT A10
COLUMN STATUS             HEADING 'Status'             FORMAT A8
COLUMN ERROR_DATE         HEADING 'Error|Date'
COLUMN ERROR_MESSAGE      HEADING 'Error Message'      FORMAT A50

SELECT DESTINATION_DBLINK,
       STATUS,
       ERROR_DATE,
       ERROR_MESSAGE
  FROM DBA_PROPAGATION
  WHERE PROPAGATION_NAME = 'STREAMS_PROPAGATION';
```

If the propagation is disabled currently, then your output looks similar to the following:

```
Database            Error
Link       Status   Date      Error Message
---------- -------- --------- --------------------------------------------------
INST2.NET  DISABLED 27-APR-05 ORA-25307: Enqueue rate too high, flow control
                              enabled
```

If there is a problem, then try the following actions to correct it:

- If a propagation is disabled, then you can enable it using the `START_PROPAGATION` procedure in the `DBMS_PROPAGATION_ADM` package, if you have not done so already.
- If the propagation is disabled or aborted, and the `Error Date` and `Error Message` fields are populated, then diagnose and correct the problem based on the error message.
- If the propagation is disabled or aborted, then check the trace file for the **propagation job** process. The query in "Displaying the Schedule for a Propagation Job" on page 21-16 displays the propagation job process.
- If the propagation job is enabled, but is not propagating messages, then try stopping and restarting the propagation.

See Also:

- "Starting a Propagation" on page 12-9
- "Checking the Trace Files and Alert Log for Problems" on page 18-21
- "Stopping a Propagation" on page 12-10
- *Oracle Database Error Messages* for more information about a specific error message

Are There Enough Job Queue Processes?

Propagation jobs use job queue processes to propagate messages. Make sure the `JOB_QUEUE_PROCESSES` initialization parameter is set to `2` or higher in each database instance that runs **propagation**s. It should be set to a value that is high enough to accommodate all of the jobs that run simultaneously.

See Also:

- "Setting Initialization Parameters Relevant to Streams" on page 10-4
- The description of propagation features in *Oracle Streams Advanced Queuing User's Guide and Reference* for more information about setting the `JOB_QUEUE_PROCESSES` initialization parameter when you use **propagation job**s
- *Oracle Database Reference* for more information about the `JOB_QUEUE_PROCESSES` initialization parameter
- *Oracle Database PL/SQL Packages and Types Reference* for more information about job queues

Is Security Configured Properly for the ANYDATA Queue?

ANYDATA queues are **secure queue**s, and security must be configured properly for users to be able to perform operations on them. If you use the SET_UP_QUEUE procedure in the DBMS_STREAMS_ADM package to configure a secure ANYDATA queue, then an error is raised if the agent that SET_UP_QUEUE tries to create already exists and is associated with a user other than the user specified by queue_user in this procedure. In this case, rename or remove the existing agent using the ALTER_AQ_AGENT or DROP_AQ_AGENT procedure, respectively, in the DBMS_AQADM package. Next, retry SET_UP_QUEUE.

In addition, you might encounter one of the following errors if security is not configured properly for an ANYDATA queue:

- ORA-24093 AQ Agent not granted privileges of database user
- ORA-25224 Sender name must be specified for enqueue into secure queues

> **See Also:** "Secure Queues" on page 3-23

ORA-24093 AQ Agent not granted privileges of database user

Secure queue access must be granted to an AQ agent explicitly for both enqueue and dequeue operations. You grant the agent these privileges using the ENABLE_DB_ACCESS procedure in the DBMS_AQADM package.

For example, to grant an agent named explicit_dq privileges of the database user oe, run the following procedure:

```
BEGIN
  DBMS_AQADM.ENABLE_DB_ACCESS(
    agent_name  => 'explicit_dq',
    db_username => 'oe');
END;
/
```

To check the privileges of the agents in a database, run the following query:

```
SELECT AGENT_NAME "Agent", DB_USERNAME "User" FROM DBA_AQ_AGENT_PRIVS;
```

Your output looks similar to the following:

```
Agent                          User
------------------------------ ------------------------------
EXPLICIT_ENQ                   OE
APPLY_OE                       OE
EXPLICIT_DQ                    OE
```

> **See Also:** "Enabling a User to Perform Operations on a Secure Queue" on page 12-3 for a detailed example that grants privileges to an agent

ORA-25224 Sender name must be specified for enqueue into secure queues

To enqueue into a secure queue, the SENDER_ID must be set to an AQ agent with secure queue privileges for the queue in the message properties.

> **See Also:** "Wrapping User Message Payloads in an ANYDATA Wrapper and Enqueuing Them" on page 12-15 for an example that sets the SENDER_ID for enqueue

Troubleshooting Apply Problems

If an **apply process** is not applying changes as expected, then use the following checklist to identify and resolve apply problems:

- Is the Apply Process Enabled?
- Is the Apply Process Current?
- Does the Apply Process Apply Captured Messages or User-Enqueued Messages?
- Is the Apply Process Queue Receiving the Messages to be Applied?
- Is a Custom Apply Handler Specified?
- Is the AQ_TM_PROCESSES Initialization Parameter Set to Zero?
- Are Any Apply Errors in the Error Queue?

See Also:

- Chapter 4, "Streams Apply Process"
- Chapter 13, "Managing an Apply Process"
- Chapter 22, "Monitoring Streams Apply Processes"

Is the Apply Process Enabled?

An **apply process** applies changes only when it is enabled. You can check whether an apply process is enabled, disabled, or aborted by querying the `DBA_APPLY` data dictionary view. For example, to check whether an apply process named `apply` is enabled, run the following query:

```
SELECT STATUS FROM DBA_APPLY WHERE APPLY_NAME = 'APPLY';
```

If the apply process is disabled, then your output looks similar to the following:

```
STATUS
--------
DISABLED
```

If the apply process is disabled, then try restarting it. If the apply process is aborted, then you might need to correct an error before you can restart it successfully.

To determine why the apply process aborted, query the `DBA_APPLY` data dictionary view or check the trace files for the apply process. The following query shows when the apply process aborted and the error that caused it to abort:

```
COLUMN APPLY_NAME HEADING 'APPLY|Process|Name' FORMAT A10
COLUMN STATUS_CHANGE_TIME HEADING 'Abort Time'
COLUMN ERROR_NUMBER HEADING 'Error Number' FORMAT 99999999
COLUMN ERROR_MESSAGE HEADING 'Error Message' FORMAT A40

SELECT APPLY_NAME, STATUS_CHANGE_TIME, ERROR_NUMBER, ERROR_MESSAGE
  FROM DBA_APPLY WHERE STATUS='ABORTED';
```

See Also:

- "Starting an Apply Process" on page 13-7
- "Displaying Detailed Information About Apply Errors" on page 22-16
- "Checking the Trace Files and Alert Log for Problems" on page 18-21
- "Streams Apply Processes and Oracle Real Application Clusters" on page 4-9 for information about restarting an apply process in an Oracle Real Application Clusters environment

Is the Apply Process Current?

If an **apply process** has not applied recent changes, then the problem might be that the apply process has fallen behind. You can check apply process latency by querying the `V$STREAMS_APPLY_COORDINATOR` dynamic performance view. If apply process latency is high, then you might be able to improve performance by adjusting the setting of the `parallelism` apply process parameter.

See Also:

- "Determining the Capture to Apply Latency for a Message for Each Apply Process" on page 22-11
- "Apply Process Parallelism" on page 4-14
- "Setting an Apply Process Parameter" on page 13-11

Does the Apply Process Apply Captured Messages or User-Enqueued Messages?

An **apply process** can apply either **captured message**s or **user-enqueued message**s, but not both types of messages. An apply process might not be applying messages of a one type because it was configured to apply the other type of messages.

You can check the type of messages applied by an apply process by querying the `DBA_APPLY` data dictionary view. For example, to check whether an apply process named `apply` applies **captured message**s or **user-enqueued message**s, run the following query:

```
COLUMN APPLY_CAPTURED HEADING 'Type of Messages Applied' FORMAT A25

SELECT DECODE(APPLY_CAPTURED,
              'YES', 'Captured',
              'NO',  'User-Enqueued') APPLY_CAPTURED
  FROM DBA_APPLY
  WHERE APPLY_NAME = 'APPLY';
```

If the apply process applies captured messages, then your output looks similar to the following:

```
Type of Messages Applied
------------------------
Captured
```

If an apply process is not applying the expected type of messages, then you might need to create a new apply process to apply the messages.

See Also:

- "Captured and User-Enqueued Messages in an ANYDATA Queue" on page 3-3
- "Creating a Capture Process" on page 11-2

Is the Apply Process Queue Receiving the Messages to be Applied?

An **apply process** must receive messages in its **queue** before it can apply these messages. Therefore, if an apply process is applying **captured message**s, then the **capture process** that captures these messages must be enabled, and it must be configured properly. Similarly, if messages are propagated from one or more databases before reaching the apply process, then each **propagation** must be enabled and must be configured properly. If a capture process or a propagation on which the apply process depends is not enabled or is not configured properly, then the messages might never reach the apply process queue.

The **rule set**s used by all **Streams client**s, including capture processes and propagations, determine the behavior of these Streams clients. Therefore, make sure the rule sets for any capture processes or propagations on which an apply process depends contain the correct **rule**s. If the **rule**s for these Streams clients are not configured properly, then the apply process queue might never receive the appropriate messages. Also, a message traveling through a stream is the composition of all of the transformations done along the path. For example, if a capture process uses **subset rule**s and performs **row migration** during capture of a message, and a propagation uses a **rule-based transformation** on the message to change the table name, then, when the message reaches an apply process, the apply process rules must account for these transformations.

In an environment where a capture process captures changes that are propagated and applied at multiple databases, you can use the following guidelines to determine whether a problem is caused by a capture process or a propagation on which an apply process depends or by the apply process itself:

- If no other **destination database**s of a capture process are applying changes from the capture process, then the problem is most likely caused by the capture process or a propagation near the capture process. In this case, first make sure the capture process is enabled and configured properly, and then make sure the propagations nearest the capture process are enabled and configured properly.
- If other destination databases of a capture process are applying changes from the capture process, then the problem is most likely caused by the apply process itself or a propagation near the apply process. In this case, first make sure the apply process is enabled and configured properly, and then make sure the propagations nearest the apply process are enabled and configured properly.

See Also:

- "Troubleshooting Capture Problems" on page 18-1
- "Troubleshooting Propagation Problems" on page 18-6
- "Troubleshooting Problems with Rules and Rule-Based Transformations" on page 18-14

Is a Custom Apply Handler Specified?

You can use **apply handler**s to handle messages dequeued by an **apply process** in a customized way. These handlers include **DML handler**s, **DDL handler**s, **precommit handler**s, and **message handler**s. If an apply process is not behaving as expected, then check the handler procedures used by the apply process, and correct any flaws. You might need to modify a handler procedure or remove it to correct an apply problem.

You can find the names of these procedures by querying the `DBA_APPLY_DML_HANDLERS` and `DBA_APPLY` data dictionary views.

See Also:

- "Message Processing Options for an Apply Process" on page 4-3 for general information about apply handlers
- Chapter 13, "Managing an Apply Process" for information about managing apply handlers
- "Displaying Information About Apply Handlers" on page 22-4 for queries that display information about apply handlers

Is the AQ_TM_PROCESSES Initialization Parameter Set to Zero?

The `AQ_TM_PROCESSES` initialization parameter controls time monitoring on **queue** messages and controls processing of messages with delay and expiration properties specified. In Oracle Database 10*g*, the database automatically controls these activities when the `AQ_TM_PROCESSES` initialization parameter is not set.

If an **apply process** is not applying messages, but there are messages that satisfy the apply process **rule set**s in the apply process queue, then make sure the `AQ_TM_PROCESSES` initialization parameter is not set to zero at the **destination database**. If this parameter is set to zero, then unset this parameter or set it to a nonzero value and monitor the apply process to see if it begins to apply messages.

To determine whether there are messages in a **buffered queue**, you can query the `V$BUFFERED_QUEUES` and `V$BUFFERED_SUBSCRIBERS` dynamic performance views. To determine whether there are **user-enqueued message**s in a queue, you can query the **queue table** for the queue.

See Also:

- "Viewing the Contents of User-Enqueued Messages in a Queue" on page 21-4
- "Monitoring Buffered Queues" on page 21-5
- *Oracle Streams Advanced Queuing User's Guide and Reference* for information about the `AQ_TM_PROCESSES` initialization parameter

Are Any Apply Errors in the Error Queue?

When an **apply process** cannot apply a message, it moves the message and all of the other messages in the same transaction into the error queue. You should check for apply errors periodically to see if there are any transactions that could not be applied.

You can check for apply errors by querying the `DBA_APPLY_ERROR` data dictionary view. Also, you can reexecute a particular transaction from the error queue or all of the transactions in the error queue.

See Also:

- "Checking for Apply Errors" on page 22-15
- "Managing Apply Errors" on page 13-23

Troubleshooting Problems with Rules and Rule-Based Transformations

When a **capture process**, a **propagation**, an **apply process**, or a **messaging client** is not behaving as expected, the problem might be that **rule**s or **rule-based transformation**s for the **Streams client** are not configured properly. Use the following checklist to identify and resolve problems with rules and rule-based transformations:

- Are Rules Configured Properly for the Streams Client?
- Are Declarative Rule-Based Transformations Configured Properly?
- Are the Custom Rule-Based Transformations Configured Properly?
- Are Incorrectly Transformed LCRs in the Error Queue?

See Also:

- Chapter 5, "Rules"
- Chapter 6, "How Rules Are Used in Streams"
- Chapter 14, "Managing Rules"

Are Rules Configured Properly for the Streams Client?

If a **capture process**, a **propagation**, an **apply process**, or a **messaging client** is behaving in an unexpected way, then the problem might be that the **rule**s in either the **positive rule set** or **negative rule set** for the **Streams client** are not configured properly. For example, if you expect a capture process to capture changes made to a particular table, but the capture process is not capturing these changes, then the cause might be that the rules in the **rule set**s used by the capture process do not instruct the capture process to capture changes to the table.

You can check the rules for a particular Streams client by querying the `DBA_STREAMS_RULES` data dictionary view. If you use both positive and negative rule sets in your Streams environment, then it is important to know whether a rule returned by this view is in the positive or negative rule set for a particular Streams client.

A Streams client performs an action, such as capture, propagation, apply, or dequeue, for messages that satisfy its rule sets. In general, a message satisfies the rule sets for a Streams client if *no rules* in the negative rule set evaluate to `TRUE` for the message, and *at least one rule* in the positive rule set evaluates to `TRUE` for the message.

"Rule Sets and Rule Evaluation of Messages" on page 6-3 contains more detailed information about how a message satisfies the rule sets for a Streams client, including information about Streams client behavior when one or more rule sets are not specified.

See Also:

- Chapter 23, "Monitoring Rules"
- "Rule Sets and Rule Evaluation of Messages" on page 6-3

This section includes the following subsections:

Checking Schema and Global Rules

Schema and **global rule**s in the positive rule set for a Streams client instruct the Streams client to perform its task for all of the messages relating to a particular schema or database, respectively. Schema and global rules in the negative rule set for a Streams client instruct the Streams client to discard all of the messages relating to a particular schema or database, respectively. If a Streams client is not behaving as expected, then it might be because schema or global rules are not configured properly for the Streams client.

For example, suppose a database is running an apply process named `strm01_apply`, and you want this apply process to apply LCRs containing changes to the `hr` schema. If the apply process uses a negative rule set, then make sure there are no **schema rule**s that evaluate to `TRUE` for this schema in the negative rule set. Such rules cause the apply process to discard LCRs containing changes to the schema. "Displaying the Rules in the Negative Rule Set for a Streams Client" on page 23-5 contains an example of a query that shows such rules.

If the query returns any such rules, then the rules returned might be causing the apply process to discard changes to the schema. If this query returns no rows, then make sure there are schema rules in the positive rule set for the apply process that evaluate to `TRUE` for the schema. "Displaying the Rules in the Positive Rule Set for a Streams Client" on page 23-4 contains an example of a query that shows such rules.

Checking Table Rules

Table rules in the positive rule set for a Streams client instruct the Streams client to perform its task for the messages relating to one or more particular tables. Table rules in the negative rule set for a Streams client instruct the Streams client to discard the messages relating to one or more particular tables.

If a Streams client is not behaving as expected for a particular table, then it might be for one of the following reasons:

- One or more global rules in the rule sets for the Streams client instruct the Streams client to behave in a particular way for messages relating to the table because the table is in a specific database. That is, a global rule in the negative rule set for the Streams client might instruct the Streams client to discard all messages from the **source database** that contains the table, or a global rule in the positive rule set for the Streams client might instruct the Streams client to perform its task for all messages from the source database that contains the table.
- One or more schema rules in the rule sets for the Streams client instruct the Streams client to behave in a particular way for messages relating to the table because the table is in a specific schema. That is, a schema rule in the negative rule set for the Streams client might instruct the Streams client to discard all messages relating to database objects in the schema, or a schema rule in the positive rule set for the Streams client might instruct the Streams client to perform its task for all messages relating to database objects in the schema.

- One or more **table rule**s in the rule sets for the Streams client instruct the Streams client to behave in a particular way for messages relating to the table.

See Also: "Checking Schema and Global Rules" on page 18-15

If you are sure that no global or schema rules are causing the unexpected behavior, then you can check for table rules in the rule sets for a Streams client. For example, if you expect a capture process to capture changes to a particular table, but the capture process is not capturing these changes, then the cause might be that the rules in the positive and negative rule sets for the capture process do not instruct it to capture changes to the table.

Suppose a database is running a capture process named `strm01_capture`, and you want this capture process to capture changes to the `hr.departments` table. If the capture process uses a negative rule set, then make sure there are no table rules that evaluate to `TRUE` for this table in the negative rule set. Such rules cause the capture process to discard changes to the table. "Displaying the Rules in the Negative Rule Set for a Streams Client" on page 23-5 contains an example of a query that shows rules in a negative rule set.

If that query returns any such rules, then the rules returned might be causing the capture process to discard changes to the table. If that query returns no rules, then make sure there are one or more table rules in the positive rule set for the capture process that evaluate to `TRUE` for the table. "Displaying the Rules in the Positive Rule Set for a Streams Client" on page 23-4 contains an example of a query that shows rules in a positive rule set.

You can also determine which rules have a particular pattern in their rule condition. "Listing Each Rule that Contains a Specified Pattern in Its Condition" on page 23-10. For example, you can find all of the rules with the string "departments" in their rule condition, and you can make sure these rules are in the correct rule sets.

See Also: "Table Rules Example" on page 6-15 for more information about specifying table rules

Checking Subset Rules

A **subset rule** can be in the rule set used by a capture process, propagation, apply process, or messaging client. A subset rule evaluates to `TRUE` only if a DML operation contains a change to a particular subset of rows in the table. For example, to check for table rules that evaluate to `TRUE` for an apply process named `strm01_apply` when there are changes to the `hr.departments` table, run the following query:

```
COLUMN RULE_NAME HEADING 'Rule Name' FORMAT A20
COLUMN RULE_TYPE HEADING 'Rule Type' FORMAT A20
COLUMN DML_CONDITION HEADING 'Subset Condition' FORMAT A30

SELECT RULE_NAME, RULE_TYPE, DML_CONDITION
  FROM DBA_STREAMS_RULES
  WHERE STREAMS_NAME   = 'STRM01_APPLY' AND
        STREAMS_TYPE   = 'APPLY' AND
        SCHEMA_NAME    = 'HR' AND
        OBJECT_NAME    = 'DEPARTMENTS';

Rule Name            Rule Type            Subset Condition
-------------------- -------------------- ------------------------------
DEPARTMENTS5         DML                  location_id=1700
DEPARTMENTS6         DML                  location_id=1700
DEPARTMENTS7         DML                  location_id=1700
```

Notice that this query returns any subset condition for the table in the DML_CONDITION column, which is labeled "Subset Condition" in the output. In this example, subset rules are specified for the hr.departments table. These subset rules evaluate to TRUE only if an LCR contains a change that involves a row where the location_id is 1700. So, if you expected the apply process to apply all changes to the table, then these subset rules cause the apply process to discard changes that involve rows where the location_id is not 1700.

Note: Subset rules must reside only in positive rule sets.

See Also:

- "Table Rules Example" on page 6-15 for more information about specifying subset rules
- "Row Migration and Subset Rules" on page 6-20

Checking for Message Rules

A message rule can be in the rule set used by a propagation, apply process, or messaging client. Message rules pertain only to **user-enqueued message**s of a specific message type, not to **captured message**s. A message rule evaluates to TRUE if a user-enqueued message in a **queue** is of the type specified in the message rule and satisfies the **rule condition** of the message rule.

If you expect a propagation, apply process, or messaging client to perform its task for some user-enqueued messages, but the Streams client is not performing its task for these messages, then the cause might be that the rules in the positive and negative rule sets for the Streams client do not instruct it to perform its task for these messages. Similarly, if you expect a propagation, apply process, or messaging client to discard some user-enqueued messages, but the Streams client is not discarding these messages, then the cause might be that the rules in the positive and negative rule sets for the Streams client do not instruct it to discard these messages.

For example, suppose you want a messaging client named oe to dequeue messages of type oe.user_msg that satisfy the following condition:

```
:"VAR$_2".OBJECT_OWNER = 'OE' AND  :"VAR$_2".OBJECT_NAME = 'ORDERS'
```

If the messaging client uses a negative rule set, then make sure there are no message rules that evaluate to TRUE for this message type in the negative rule set. Such rules cause the messaging client to discard these messages. For example, to determine whether there are any such rules in the negative rule set for the messaging client, run the following query:

```
COLUMN RULE_NAME HEADING 'Rule Name' FORMAT A30
COLUMN RULE_CONDITION HEADING 'Rule Condition' FORMAT A30

SELECT RULE_NAME, RULE_CONDITION
  FROM DBA_STREAMS_RULES
  WHERE STREAMS_NAME       = 'OE' AND
        MESSAGE_TYPE_OWNER = 'OE' AND
        MESSAGE_TYPE_NAME  = 'USER_MSG' AND
        RULE_SET_TYPE      = 'NEGATIVE';
```

If this query returns any rules, then the rules returned might be causing the messaging client to discard messages. Examine the rule condition of the returned rules to determine whether these rules are causing the messaging client to discard the messages that it should be dequeuing. If this query returns no rules, then make sure there are message rules in the positive rule set for the messaging client that evaluate to TRUE for this message type and condition.

For example, to determine whether any message rules evaluate to TRUE for this message type in the positive rule set for the messaging client, run the following query:

```
COLUMN RULE_NAME HEADING 'Rule Name' FORMAT A35
COLUMN RULE_CONDITION HEADING 'Rule Condition' FORMAT A35

SELECT RULE_NAME, RULE_CONDITION
  FROM DBA_STREAMS_RULES
  WHERE STREAMS_NAME       = 'OE' AND
        MESSAGE_TYPE_OWNER = 'OE' AND
        MESSAGE_TYPE_NAME  = 'USER_MSG' AND
        RULE_SET_TYPE      = 'POSITIVE';
```

If you have message rules that evaluate to TRUE for this message type in the positive rule set for the messaging client, then these rules are returned. In this case, your output looks similar to the following:

```
Rule Name                          Rule Condition
---------------------------------- ----------------------------------
RULE$_3                            :"VAR$_2".OBJECT_OWNER = 'OE' AND
                                   :"VAR$_2".OBJECT_NAME = 'ORDERS'
```

Examine the rule condition for the rules returned to determine whether they instruct the messaging client to dequeue the proper messages. Based on these results, the messaging client named oe should dequeue messages of oe.user_msg type that satisfy condition shown in the output. In other words, no rule in the negative messaging client rule set discards these messages, and a rule exists in the positive messaging client rule set that evaluates to TRUE when the messaging client finds a message in its queue of the of oe.user_msg type that satisfies the rule condition.

See Also:

- "Message Rule Example" on page 6-27 for more information about specifying message rules
- "Configuring a Messaging Client and Message Notification" on page 12-18 for an example that creates the rule discussed in this section

Resolving Problems with Rules

If you determine that a Streams capture process, propagation, apply process, or messaging client is not behaving as expected because one or more rules must be added to the rule set for the Streams client, then you can use one of the following procedures in the DBMS_STREAMS_ADM package to add appropriate rules:

- ADD_GLOBAL_PROPAGATION_RULES
- ADD_GLOBAL_RULES
- ADD_SCHEMA_PROPAGATION_RULES
- ADD_SCHEMA_RULES
- ADD_SUBSET_PROPAGATION_RULES

- ADD_SUBSET_RULES
- ADD_TABLE_PROPAGATION_RULES
- ADD_TABLE_RULES
- ADD_MESSAGE_PROPAGATION_RULE
- ADD_MESSAGE_RULE

You can use the DBMS_RULE_ADM package to add customized rules, if necessary.

It is also possible that the Streams capture process, propagation, apply process, or messaging client is not behaving as expected because one or more rules should be altered or removed from a rule set.

If you have the correct rules, and the relevant messages are still filtered out by a Streams capture process, propagation, or apply process, then check your trace files and alert log for a warning about a missing "multi-version data dictionary", which is a **Streams data dictionary**. The following information might be included in such warning messages:

- gdbnm: Global name of the source database of the missing object
- scn: SCN for the transaction that has been missed

If you find such messages, and you are using custom capture process rules or reusing existing capture process rules for a new **destination database**, then make sure you run the appropriate procedure to prepare for **instantiation**:

- PREPARE_TABLE_INSTANTIATION
- PREPARE_SCHEMA_INSTANTIATION
- PREPARE_GLOBAL_INSTANTIATION

Also, make sure propagation is working from the source database to the destination database. Streams data dictionary information is propagated to the destination database and loaded into the dictionary at the destination database.

See Also:

- "Altering a Rule" on page 14-6
- "Removing a Rule from a Rule Set" on page 14-3
- *Oracle Streams Replication Administrator's Guide* for more information about preparing database objects for instantiation
- "The Streams Data Dictionary" on page 2-37 for more information about the Streams data dictionary

Are Declarative Rule-Based Transformations Configured Properly?

A **declarative rule-based transformation** is a rule-based transformation that covers one of a common set of transformation scenarios for row LCRs. Declarative rule-based transformations are run internally without using PL/SQL. If a Streams **capture process**, **propagation**, **apply process**, or **messaging client** is not behaving as expected, then check the declarative rule-based transformations specified for the **rule**s used by the Streams client and correct any mistakes.

The most common problems with declarative rule-based transformations are:

- The declarative rule-based transformation is specified for a table or involves columns in a table, but the schema either was not specified or was incorrectly specified when the transformation was created. If the schema is not correct in a declarative rule-based transformation, then the transformation will not be run on the appropriate LCRs. You should specify the owning schema for a table when you create a declarative rule-based transformation. If the schema is not specified when a declarative rule-based transformation is created, then the user who creates the transformation is specified for the schema by default.

  If the schema is not correct for a declarative rule-based transformation, then, to correct the problem, remove the transformation and re-create it, specifying the correct schema for each table.

- If more than one declarative rule-based transformation is specified for a particular rule, then make sure the ordering is correct for execution of these transformations. Incorrect ordering of declarative rule-based transformations can result in errors or inconsistent data.

  If the ordering is not correct for the declarative rule-based transformation specified on a single rule, then, to correct the problem, remove the transformations and re-create them with the correct ordering.

See Also:

- "Displaying Declarative Rule-Based Transformations" on page 24-2
- "Transformation Ordering" on page 7-12

Are the Custom Rule-Based Transformations Configured Properly?

A **custom rule-based transformation** is any modification by a user-defined function to a message when a **rule** evaluates to `TRUE`. A custom rule-based transformation is specified in the **action context** of a rule, and these action contexts contain a name-value pair with `STREAMS$_TRANSFORM_FUNCTION` for the name and a user-created function name for the value. This user-created function performs the transformation. If the user-created function contains any flaws, then unexpected behavior can result.

If a Streams **capture process**, **propagation**, **apply process**, or **messaging client** is not behaving as expected, then check the custom rule-based transformation functions specified for the rules used by the **Streams client** and correct any flaws. You can find the names of these functions by querying the `DBA_STREAMS_TRANSFORM_FUNCTION` data dictionary view. You might need to modify a transformation function or remove a custom rule-based transformation to correct the problem. Also, make sure the name of the function is spelled correctly when you specify the transformation for a rule.

An error caused by a custom rule-based transformation might cause a capture process, propagation, apply process, or messaging client to abort. In this case, you might need to correct the transformation before the Streams client can be restarted or invoked.

Rule evaluation is done before a custom rule-based transformation. For example, if you have a transformation that changes the name of a table from `emps` to `employees`, then make sure each rule using the transformation specifies the table name `emps`, rather than `employees`, in its **rule condition**.

See Also:

- "Displaying Custom Rule-Based Transformations" on page 24-5 for a query that displays the custom rule-based transformation functions specified for the rules in a **rule set**
- "Managing Custom Rule-Based Transformations" on page 15-5 for information about modifying or removing custom rule-based transformations

Are Incorrectly Transformed LCRs in the Error Queue?

In some cases, incorrectly transformed LCRs might have been moved to the error queue by an **apply process**. When this occurs, you should examine the transaction in the error queue to analyze the feasibility of reexecuting the transaction successfully. If an abnormality is found in the transaction, then you might be able to configure a **DML handler** to correct the problem. The DML handler will run when you reexecute the error transaction. When a DML handler is used to correct a problem in an error transaction, the apply process that uses the DML handler should be stopped to prevent the DML handler from acting on LCRs that are not involved with the error transaction. After successful reexecution, if the DML handler is no longer needed, then remove it. Also, correct the **rule-based transformation** to avoid future errors.

See Also:

- "The Error Queue" on page 4-16
- "Displaying Detailed Information About Apply Errors" on page 22-16

Checking the Trace Files and Alert Log for Problems

Messages about each **capture process**, **propagation**, and **apply process** are recorded in trace files for the database in which the process or **propagation job** is running. A **local capture process** runs on a **source database**, a **downstream capture process** runs on a **downstream database**, a propagation job runs on the database containing the **source queue** in the propagation, and an apply process runs on a **destination database**. These trace file messages can help you to identify and resolve problems in a Streams environment.

All trace files for background processes are written to the destination directory specified by the initialization parameter `BACKGROUND_DUMP_DEST`. The names of trace files are operating system specific, but each file usually includes the name of the process writing the file.

For example, on some operating systems, the trace file name for a process is *`sid`*`_`*`xxxxx`*`_`*`iiiii`*`.trc`, where:

- *`sid`* is the system identifier for the database
- *`xxxxx`* is the name of the process
- *`iiiii`* is the operating system process number

Also, you can set the `write_alert_log` parameter to `y` for both a capture process and an apply process. When this parameter is set to `y`, which is the default setting, the alert log for the database contains messages about why the capture process or apply process stopped.

You can control the information in the trace files by setting the `trace_level` capture process or apply process parameter using the `SET_PARAMETER` procedure in the `DBMS_CAPTURE_ADM` and `DBMS_APPLY_ADM` packages.

Use the following checklist to check the trace files related to Streams:

- Does a Capture Process Trace File Contain Messages About Capture Problems?
- Do the Trace Files Related to Propagation Jobs Contain Messages About Problems?
- Does an Apply Process Trace File Contain Messages About Apply Problems?

See Also:

- *Oracle Database Administrator's Guide* for more information about trace files and the alert log, and for more information about their names and locations
- *Oracle Database PL/SQL Packages and Types Reference* for more information about setting the `trace_level` capture process parameter and the `trace_level` apply process parameter
- Your operating system specific Oracle documentation for more information about the names and locations of trace files

Does a Capture Process Trace File Contain Messages About Capture Problems?

A capture process is an Oracle background process named `c`*nnn*, where *nnn* is the capture process number. For example, on some operating systems, if the system identifier for a database running a capture process is `hqdb` and the capture process number is `01`, then the trace file for the capture process starts with `hqdb_c001`.

See Also: "Displaying Change Capture Information About Each Capture Process" on page 20-3 for a query that displays the capture process number of a capture process

Do the Trace Files Related to Propagation Jobs Contain Messages About Problems?

Each propagation uses a propagation job that depends on the job queue coordinator process and a job queue process. The job queue coordinator process is named `cjq`*nn*, where *nn* is the job queue coordinator process number, and a job queue process is named `j`*nnn*, where *nnn* is the job queue process number.

For example, on some operating systems, if the system identifier for a database running a propagation job is `hqdb` and the job queue coordinator process is `01`, then the trace file for the job queue coordinator process starts with `hqdb_cjq01`. Similarly, on the same database, if a job queue process is `001`, then the trace file for the job queue process starts with `hqdb_j001`. You can check the process name by querying the `PROCESS_NAME` column in the `DBA_QUEUE_SCHEDULES` data dictionary view.

See Also: "Is the Propagation Enabled?" on page 18-7 for a query that displays the job queue process used by a propagation job

Does an Apply Process Trace File Contain Messages About Apply Problems?

An **apply process** is an Oracle background process named `a`*`nnn`*, where *`nnn`* is the apply process number. For example, on some operating systems, if the system identifier for a database running an apply process is `hqdb` and the apply process number is `001`, then the trace file for the apply process starts with `hqdb_a001`.

An apply process also uses parallel execution servers. Information about an apply process might be recorded in the trace file for one or more parallel execution servers. The process name of a parallel execution server is `p`*`nnn`*, where *`nnn`* is the process number. So, on some operating systems, if the system identifier for a database running an apply process is `hqdb` and the process number is `001`, then the trace file that contains information about a parallel execution server used by an apply process starts with `hqdb_p001`.

See Also:

- "Displaying General Information About Each Coordinator Process" on page 22-9 for a query that displays the apply process number of an apply process
- "Displaying Information About the Reader Server for Each Apply Process" on page 22-6 for a query that displays the parallel execution server used by the **reader server** of an apply process
- "Displaying Information About the Apply Servers for Each Apply Process" on page 22-9 for a query that displays the parallel execution servers used by the **apply servers** of an apply process

Part III

Monitoring Streams

This part describes monitoring a Streams environment. This part contains the following chapters:

19

Monitoring a Streams Environment

This chapter lists the static data dictionary views and dynamic performance views related to Streams. You can use these views to monitor your Streams environment.

This chapter contains these topics:

- Summary of Streams Static Data Dictionary Views
- Summary of Streams Dynamic Performance Views

Note: The Streams tool in the Oracle Enterprise Manager Console is also an excellent way to monitor a Streams environment. See the online help for the Streams tool for more information.

See Also:

- *Oracle Database Reference* for information about the data dictionary views described in this chapter
- *Oracle Streams Replication Administrator's Guide* for information about monitoring a Streams **replication** environment

Summary of Streams Static Data Dictionary Views

Table 19–1 lists the Streams static data dictionary views.

Table 19–1 Streams Static Data Dictionary Views

ALL_ Views	DBA_ Views	USER_ Views
ALL_APPLY	DBA_APPLY	N/A
ALL_APPLY_CONFLICT_COLUMNS	DBA_APPLY_CONFLICT_COLUMNS	N/A
ALL_APPLY_DML_HANDLERS	DBA_APPLY_DML_HANDLERS	N/A
ALL_APPLY_ENQUEUE	DBA_APPLY_ENQUEUE	N/A
ALL_APPLY_ERROR	DBA_APPLY_ERROR	N/A
ALL_APPLY_EXECUTE	DBA_APPLY_EXECUTE	N/A
N/A	DBA_APPLY_INSTANTIATED_GLOBAL	N/A
N/A	DBA_APPLY_INSTANTIATED_OBJECTS	N/A
N/A	DBA_APPLY_INSTANTIATED_SCHEMAS	N/A
ALL_APPLY_KEY_COLUMNS	DBA_APPLY_KEY_COLUMNS	N/A
N/A	DBA_APPLY_OBJECT_DEPENDENCIES	N/A
ALL_APPLY_PARAMETERS	DBA_APPLY_PARAMETERS	N/A
ALL_APPLY_PROGRESS	DBA_APPLY_PROGRESS	N/A
N/A	DBA_APPLY_SPILL_TXN	N/A
ALL_APPLY_TABLE_COLUMNS	DBA_APPLY_TABLE_COLUMNS	N/A
N/A	DBA_APPLY_VALUE_DEPENDENCIES	N/A
ALL_CAPTURE	DBA_CAPTURE	N/A
ALL_CAPTURE_EXTRA_ATTRIBUTES	DBA_CAPTURE_EXTRA_ATTRIBUTES	N/A
ALL_CAPTURE_PARAMETERS	DBA_CAPTURE_PARAMETERS	N/A
ALL_CAPTURE_PREPARED_DATABASE	DBA_CAPTURE_PREPARED_DATABASE	N/A
ALL_CAPTURE_PREPARED_SCHEMAS	DBA_CAPTURE_PREPARED_SCHEMAS	N/A
ALL_CAPTURE_PREPARED_TABLES	DBA_CAPTURE_PREPARED_TABLES	N/A
ALL_EVALUATION_CONTEXT_TABLES	DBA_EVALUATION_CONTEXT_TABLES	USER_EVALUATION_CONTEXT_TABLES
ALL_EVALUATION_CONTEXT_VARS	DBA_EVALUATION_CONTEXT_VARS	USER_EVALUATION_CONTEXT_VARS
ALL_EVALUATION_CONTEXTS	DBA_EVALUATION_CONTEXTS	USER_EVALUATION_CONTEXTS
ALL_FILE_GROUP_EXPORT_INFO	DBA_FILE_GROUP_EXPORT_INFO	USER_FILE_GROUP_EXPORT_INFO
ALL_FILE_GROUP_FILES	DBA_FILE_GROUP_FILES	USER_FILE_GROUP_FILES
ALL_FILE_GROUP_TABLES	DBA_FILE_GROUP_TABLES	USER_FILE_GROUP_TABLES
ALL_FILE_GROUP_TABLESPACES	DBA_FILE_GROUP_TABLESPACES	USER_FILE_GROUP_TABLESPACES
ALL_FILE_GROUP_VERSIONS	DBA_FILE_GROUP_VERSIONS	USER_FILE_GROUP_VERSIONS
ALL_FILE_GROUPS	DBA_FILE_GROUPS	USER_FILE_GROUPS
N/A	DBA_HIST_STREAMS_APPLY_SUM	N/A
N/A	DBA_HIST_STREAMS_CAPTURE	N/A
N/A	DBA_HIST_STREAMS_POOL_ADVICE	N/A
ALL_PROPAGATION	DBA_PROPAGATION	N/A
N/A	DBA_REGISTERED_ARCHIVED_LOG	N/A
ALL_RULE_SET_RULES	DBA_RULE_SET_RULES	USER_RULE_SET_RULES
ALL_RULE_SETS	DBA_RULE_SETS	USER_RULE_SETS
ALL_RULES	DBA_RULES	USER_RULES

Table 19–1 (Cont.) Streams Static Data Dictionary Views

ALL_ Views	DBA_ Views	USER_ Views
N/A	DBA_STREAMS_ADD_COLUMN	N/A
N/A	DBA_STREAMS_ADMINISTRATOR	N/A
N/A	DBA_STREAMS_DELETE_COLUMN	N/A
ALL_STREAMS_GLOBAL_RULES	DBA_STREAMS_GLOBAL_RULES	N/A
ALL_STREAMS_MESSAGE_CONSUMERS	DBA_STREAMS_MESSAGE_CONSUMERS	N/A
ALL_STREAMS_MESSAGE_RULES	DBA_STREAMS_MESSAGE_RULES	N/A
ALL_STREAMS_NEWLY_SUPPORTED	DBA_STREAMS_NEWLY_SUPPORTED	N/A
N/A	DBA_STREAMS_RENAME_COLUMN	N/A
N/A	DBA_STREAMS_RENAME_SCHEMA	N/A
N/A	DBA_STREAMS_RENAME_TABLE	N/A
ALL_STREAMS_RULES	DBA_STREAMS_RULES	N/A
ALL_STREAMS_SCHEMA_RULES	DBA_STREAMS_SCHEMA_RULES	N/A
ALL_STREAMS_TABLE_RULES	DBA_STREAMS_TABLE_RULES	N/A
ALL_STREAMS_TRANSFORM_FUNCTION	DBA_STREAMS_TRANSFORM_FUNCTION	N/A
N/A	DBA_STREAMS_TRANSFORMATIONS	N/A
ALL_STREAMS_UNSUPPORTED	DBA_STREAMS_UNSUPPORTED	N/A

Summary of Streams Dynamic Performance Views

The Streams dynamic performance views are:

- V$BUFFERED_PUBLISHERS
- V$BUFFERED_QUEUES
- V$BUFFERED_SUBSCRIBERS
- V$PROPAGATION_RECEIVER
- V$PROPAGATION_SENDER
- V$RULE
- V$RULE_SET
- V$RULE_SET_AGGREGATE_STATS
- V$STREAMS_APPLY_COORDINATOR
- V$STREAMS_APPLY_READER
- V$STREAMS_APPLY_SERVER
- V$STREAMS_CAPTURE
- V$STREAMS_POOL_ADVICE
- V$STREAMS_TRANSACTION

Note: To collect elapsed time statistics in these dynamic performance views, set the TIMED_STATISTICS initialization parameter to true.

20

Monitoring Streams Capture Processes

This chapter provides sample queries that you can use to monitor Streams environment **capture process**es.

This chapter contains these topics:

Note: The Streams tool in the Oracle Enterprise Manager Console is also an excellent way to monitor a Streams environment. See the online help for the Streams tool for more information.

See Also:

- Chapter 2, "Streams Capture Process"
- Chapter 11, "Managing a Capture Process"
- *Oracle Database Reference* for information about the data dictionary views described in this chapter
- *Oracle Streams Replication Administrator's Guide* for information about monitoring a Streams **replication** environment

Displaying the Queue, Rule Sets, and Status of Each Capture Process

You can display the following information about each **capture process** in a database by running the query in this section:

- The capture process name
- The name of the **queue** used by the capture process
- The name of the **positive rule set** used by the capture process
- The name of the **negative rule set** used by the capture process
- The status of the capture process, which can be ENABLED, DISABLED, or ABORTED

To display this general information about each capture process in a database, run the following query:

```
COLUMN CAPTURE_NAME HEADING 'Capture|Process|Name' FORMAT A15
COLUMN QUEUE_NAME HEADING 'Capture|Process|Queue' FORMAT A15
COLUMN RULE_SET_NAME HEADING 'Positive|Rule Set' FORMAT A15
COLUMN NEGATIVE_RULE_SET_NAME HEADING 'Negative|Rule Set' FORMAT A15
COLUMN STATUS HEADING 'Capture|Process|Status' FORMAT A15

SELECT CAPTURE_NAME, QUEUE_NAME, RULE_SET_NAME, NEGATIVE_RULE_SET_NAME, STATUS
   FROM DBA_CAPTURE;
```

Your output looks similar to the following:

```
Capture         Capture                                        Capture
Process         Process         Positive        Negative       Process
Name            Queue           Rule Set        Rule Set       Status
--------------- --------------- --------------- --------------- ---------------
STRM01_CAPTURE  STRM01_QUEUE    RULESET$_25     RULESET$_36     ENABLED
```

If the status of a capture process is ABORTED, then you can query the ERROR_NUMBER and ERROR_MESSAGE columns in the DBA_CAPTURE data dictionary view to determine the error.

See Also: "Is the Capture Process Enabled?" on page 18-2 for an example query that shows the error number and error message if a capture process is aborted

Displaying Change Capture Information About Each Capture Process

The query in this section displays the following information about each **capture process** in a database:

- The name of the capture process.
- The process number (*cnnn*).
- The session identifier.
- The serial number of the session.
- The current state of the capture process:
  - INITIALIZING
  - WAITING FOR DICTONARY REDO
  - DICTIONARY INITIALIZATION
  - MINING
  - LOADING
  - CAPTURING CHANGES
  - WAITING FOR REDO
  - EVALUATING RULE
  - CREATING LCR
  - ENQUEUING MESSAGE
  - PAUSED FOR FLOW CONTROL
  - SHUTTING DOWN
- The total number of redo entries passed by LogMiner to the capture process for detailed **rule** evaluation. A capture process converts a redo entry into a **message** and performs detailed rule evaluation on the message when capture process prefiltering cannot discard the change.
- The total number LCRs enqueued since the capture process was last started.

To display this information for each capture process in a database, run the following query:

```
COLUMN CAPTURE_NAME HEADING 'Capture|Name' FORMAT A7
COLUMN PROCESS_NAME HEADING 'Capture|Process|Number' FORMAT A7
COLUMN SID HEADING 'Session|ID' FORMAT 9999
COLUMN SERIAL# HEADING 'Session|Serial|Number' FORMAT 9999
COLUMN STATE HEADING 'State' FORMAT A27
COLUMN TOTAL_MESSAGES_CAPTURED HEADING 'Redo|Entries|Evaluated|In Detail' FORMAT
9999999
COLUMN TOTAL_MESSAGES_ENQUEUED HEADING 'Total|LCRs|Enqueued' FORMAT 999999

SELECT c.CAPTURE_NAME,
       SUBSTR(s.PROGRAM,INSTR(s.PROGRAM,'(')+1,4) PROCESS_NAME,
       c.SID,
       c.SERIAL#,
       c.STATE,
       c.TOTAL_MESSAGES_CAPTURED,
       c.TOTAL_MESSAGES_ENQUEUED
  FROM V$STREAMS_CAPTURE c, V$SESSION s
  WHERE c.SID = s.SID AND
        c.SERIAL# = s.SERIAL#;
```

Your output looks similar to the following:

```
                                                                 Redo
        Capture         Session                                Entries    Total
Capture Process Session  Serial                               Evaluated     LCRs
Name    Number       ID  Number State                         In Detail Enqueued
------- ------- ------- ------- --------------------------- --------- --------
CAPTURE C001        964       3 CAPTURING CHANGES              189346      565
```

The number of redo entries scanned can be higher than the number of DML and DDL redo entries captured by a capture process. Only DML and DDL redo entries that satisfy the **rule set**s of a capture process are captured and enqueued into the capture process **queue**. Also, the total LCRs enqueued includes LCRs that contain transaction control statements. These row LCRs contain directives such as `COMMIT` and `ROLLBACK`. Therefore, the total LCRs enqueued is a number higher than the number of row changes and DDL changes enqueued by a capture process.

See Also:

- "Row LCRs" on page 2-3 for more information about transaction control statements
- "Capture Process States" on page 2-24

Displaying State Change and Message Creation Time for Each Capture Process

The query in this section displays the following information for each **capture process** in a database:

- The name of the capture process
- The current state of the capture process:
  - `INITIALIZING`
  - `WAITING FOR DICTONARY REDO`
  - `DICTIONARY INITIALIZATION`
  - `MINING`
  - `LOADING`
  - `CAPTURING CHANGES`
  - `WAITING FOR REDO`
  - `EVALUATING RULE`
  - `CREATING LCR`
  - `ENQUEUING MESSAGE`
  - `PAUSED FOR FLOW CONTROL`
  - `SHUTTING DOWN`
- The date and time when the capture process state last changed
- The date and time when the capture process last created an LCR

To display this information for each capture process in a database, run the following query:

```
COLUMN CAPTURE_NAME HEADING 'Capture|Name' FORMAT A15
COLUMN STATE HEADING 'State' FORMAT A27
COLUMN STATE_CHANGED HEADING 'State|Change Time'
COLUMN CREATE_MESSAGE HEADING 'Last Message|Create Time'

SELECT CAPTURE_NAME,
       STATE,
       TO_CHAR(STATE_CHANGED_TIME, 'HH24:MI:SS MM/DD/YY') STATE_CHANGED,
       TO_CHAR(CAPTURE_MESSAGE_CREATE_TIME, 'HH24:MI:SS MM/DD/YY') CREATE_MESSAGE
  FROM V$STREAMS_CAPTURE;
```

Your output looks similar to the following:

```
Capture                                      State             Last Message
Name            State                        Change Time       Create Time
--------------- ---------------------------- ----------------- -----------------
CAPTURE_SIMP    CAPTURING CHANGES            13:24:42 11/08/04 13:24:41 11/08/04
```

See Also: "Capture Process States" on page 2-24

Displaying Elapsed Time Performing Capture Operations for Each Capture Process

The query in this section displays the following information for each **capture process** in a database:

- The name of the capture process
- The elapsed capture time, which is the amount of time (in seconds) spent scanning for changes in the redo log since the capture process was last started
- The elapsed **rule** evaluation time, which is the amount of time (in seconds) spent evaluating rules since the capture process was last started
- The elapsed enqueue time, which is the amount of time (in seconds) spent enqueuing **message**s since the capture process was last started
- The elapsed LCR creation time, which is the amount of time (in seconds) spent creating logical change records (LCRs) since the capture process was last started
- The elapsed pause time, which is the amount of time (in seconds) spent paused for flow control since the capture process was last started

Note: All times for this query are displayed in seconds. The `V$STREAMS_CAPTURE` view displays elapsed time in centiseconds by default. A centisecond is one-hundredth of a second. The query in this section divides each elapsed time by one hundred to display the elapsed time in seconds.

To display this information for each capture process in a database, run the following query:

```
COLUMN CAPTURE_NAME HEADING 'Capture|Name' FORMAT A15
COLUMN ELAPSED_CAPTURE_TIME HEADING 'Elapsed|Capture|Time' FORMAT 99999999.99
COLUMN ELAPSED_RULE_TIME HEADING 'Elapsed|Rule|Evaluation|Time' FORMAT 99999999.99
COLUMN ELAPSED_ENQUEUE_TIME HEADING 'Elapsed|Enqueue|Time' FORMAT 99999999.99
COLUMN ELAPSED_LCR_TIME HEADING 'Elapsed|LCR|Creation|Time' FORMAT 99999999.99
COLUMN ELAPSED_PAUSE_TIME HEADING 'Elapsed|Pause|Time' FORMAT 99999999.99

SELECT CAPTURE_NAME,
       (ELAPSED_CAPTURE_TIME/100) ELAPSED_CAPTURE_TIME,
       (ELAPSED_RULE_TIME/100) ELAPSED_RULE_TIME,
       (ELAPSED_ENQUEUE_TIME/100) ELAPSED_ENQUEUE_TIME,
       (ELAPSED_LCR_TIME/100) ELAPSED_LCR_TIME,
       (ELAPSED_PAUSE_TIME/100) ELAPSED_PAUSE_TIME
  FROM V$STREAMS_CAPTURE;
```

Your output looks similar to the following:

```
                               Elapsed                    Elapsed
                    Elapsed      Rule      Elapsed          LCR      Elapsed
Capture             Capture   Evaluation   Enqueue      Creation       Pause
Name                   Time       Time        Time          Time        Time
--------------- ------------ ---------- ---------- ------------ ----------
STM1$CAP             1213.92        .04      33.84       185.25      600.60
```

Displaying Information About Each Downstream Capture Process

A downstream capture is a **capture process** runs on a database other than the **source database**. You can display the following information about each **downstream capture process** in a database by running the query in this section:

- The capture process name
- The **source database** of the changes captured by the capture process
- The name of the **queue** used by the capture process
- The status of the capture process, which can be `ENABLED`, `DISABLED`, or `ABORTED`
- Whether the downstream capture process uses a database link to the source database for administrative actions

To display this information about each downstream capture process in a database, run the following query:

```
COLUMN CAPTURE_NAME HEADING 'Capture|Process|Name' FORMAT A15
COLUMN SOURCE_DATABASE HEADING 'Source|Database' FORMAT A15
COLUMN QUEUE_NAME HEADING 'Capture|Process|Queue' FORMAT A15
COLUMN STATUS HEADING 'Capture|Process|Status' FORMAT A15
COLUMN USE_DATABASE_LINK HEADING 'Uses|Database|Link?' FORMAT A8

SELECT CAPTURE_NAME,
       SOURCE_DATABASE,
       QUEUE_NAME,
       STATUS,
       USE_DATABASE_LINK
   FROM DBA_CAPTURE
   WHERE CAPTURE_TYPE = 'DOWNSTREAM';
```

Your output looks similar to the following:

```
Capture                        Capture         Capture         Uses
Process         Source         Process         Process         Database
Name            Database       Queue           Status          Link?
--------------- -------------- --------------- --------------- --------
STRM03_CAPTURE  DBS1.NET        STRM03_QUEUE    ENABLED         YES
```

In this case, the source database for the capture process is `dbs1.net`, but the local database running the capture process is not `dbs1.net`. Also, the capture process returned by this query uses a database link to the source database to perform administrative actions. The database link name is the same as the global name of the source database, which is `dbs1.net` in this case.

If the status of a capture process is `ABORTED`, then you can query the `ERROR_NUMBER` and `ERROR_MESSAGE` columns in the `DBA_CAPTURE` data dictionary view to determine the error.

See Also:

- "Local Capture and Downstream Capture" on page 2-12
- "Creating an Archived-Log Downstream Capture Process that Assigns Logs Implicitly" on page 11-13
- "Is the Capture Process Enabled?" on page 18-2 for an example query that shows the error number and error message if a capture process is aborted

Displaying the Registered Redo Log Files for Each Capture Process

You can display information about the archived redo log files that are registered for each **capture process** in a database by running the query in this section. This query displays information about these files for both **local capture process**es and **downstream capture process**es.

The query displays the following information for each registered archived redo log file:

- The name of a capture process that uses the file
- The **source database** of the file
- The sequence number of the file
- The name and location of the file at the local site
- Whether the file contains the beginning of a data dictionary build
- Whether the file contains the end of a data dictionary build

To display this information about each registered archive redo log file in a database, run the following query:

```
COLUMN CONSUMER_NAME HEADING 'Capture|Process|Name' FORMAT A15
COLUMN SOURCE_DATABASE HEADING 'Source|Database' FORMAT A10
COLUMN SEQUENCE# HEADING 'Sequence|Number' FORMAT 99999
COLUMN NAME HEADING 'Archived Redo Log|File Name' FORMAT A20
COLUMN DICTIONARY_BEGIN HEADING 'Dictionary|Build|Begin' FORMAT A10
COLUMN DICTIONARY_END HEADING 'Dictionary|Build|End' FORMAT A10
```

```
SELECT r.CONSUMER_NAME,
       r.SOURCE_DATABASE,
       r.SEQUENCE#,
       r.NAME,
       r.DICTIONARY_BEGIN,
       r.DICTIONARY_END
  FROM DBA_REGISTERED_ARCHIVED_LOG r, DBA_CAPTURE c
  WHERE r.CONSUMER_NAME = c.CAPTURE_NAME;
```

Your output looks similar to the following:

```
Capture                                                    Dictionary Dictionary
Process         Source     Sequence Archived Redo Log      Build      Build
Name            Database     Number File Name              Begin      End
--------------- ---------- -------- -------------------- ---------- ----------
STRM02_CAPTURE  DBS2.NET         15 /orc/dbs/log/arch2_1 NO         NO
                                       _15_478347508.arc
STRM02_CAPTURE  DBS2.NET         16 /orc/dbs/log/arch2_1 NO         NO
                                       _16_478347508.arc
STRM03_CAPTURE  DBS1.NET         45 /remote_logs/arch1_1 YES        YES
                                       _45_478347335.arc
STRM03_CAPTURE  DBS1.NET         46 /remote_logs/arch1_1 NO         NO
                                       _46_478347335.arc
STRM03_CAPTURE  DBS1.NET         47 /remote_logs/arch1_1 NO         NO
                                       _47_478347335.arc
```

Assume that this query was run at the `dbs2.net` database, and that `strm02_capture` is a **local capture process**, and `strm03_capture` is a **downstream capture process**. The **source database** for the `strm03_capture` downstream capture process is `dbs1.net`. This query shows that there are two registered archived redo log files for `strm02_capture` and three registered archived redo log files for `strm02_capture`. This query shows the name and location of each of these files in the local file system.

See Also:

- "The LogMiner Data Dictionary for a Capture Process" on page 2-28 for more information about data dictionary builds
- "Local Capture and Downstream Capture" on page 2-12
- "Creating an Archived-Log Downstream Capture Process that Assigns Logs Implicitly" on page 11-13
- "ARCHIVELOG Mode and a Capture Process" on page 2-38

Displaying the Redo Log Files that Are Required by Each Capture Process

A **capture process** needs the redo log file that includes the **required checkpoint SCN**, and all subsequent redo log files. You can query the `REQUIRED_CHECKPOINT_SCN` column in the `DBA_CAPTURE` data dictionary view to determine the required checkpoint SCN for a capture process. Redo log files prior to the redo log file that contains the required checkpoint SCN are no longer needed by the capture process. These redo log files can be stored offline if they are no longer needed for any other purpose. If you reset the **start SCN** for a capture process to a lower value in the future, then these redo log files might be needed.

The query displays the following information for each required archived redo log file:

- The name of a capture process that uses the file
- The **source database** of the file
- The sequence number of the file
- The name and location of the required redo log file at the local site

To display this information about each required archive redo log file in a database, run the following query:

```
COLUMN CONSUMER_NAME HEADING 'Capture|Process|Name' FORMAT A15
COLUMN SOURCE_DATABASE HEADING 'Source|Database' FORMAT A10
COLUMN SEQUENCE# HEADING 'Sequence|Number' FORMAT 99999
COLUMN NAME HEADING 'Required|Archived Redo Log|File Name' FORMAT A40

SELECT r.CONSUMER_NAME,
       r.SOURCE_DATABASE,
       r.SEQUENCE#,
       r.NAME
  FROM DBA_REGISTERED_ARCHIVED_LOG r, DBA_CAPTURE c
  WHERE r.CONSUMER_NAME =  c.CAPTURE_NAME AND
        r.NEXT_SCN      >= c.REQUIRED_CHECKPOINT_SCN;
```

Your output looks similar to the following:

```
Capture                              Required
Process         Source     Sequence Archived Redo Log
Name            Database     Number File Name
--------------- ---------- -------- ----------------------------------------
STRM02_CAPTURE  DBS2.NET         16 /orc/dbs/log/arch2_1_16_478347508.arc
STRM03_CAPTURE  DBS1.NET         47 /remote_logs/arch1_1_47_478347335.arc
```

See Also: "Required Checkpoint SCN" on page 2-25

Displaying SCN Values for Each Redo Log File Used by Each Capture Process

You can display information about the SCN values for archived redo log files that are registered for each **capture process** in a database by running the query in this section. This query displays information the SCN values for these files for both **local capture process**es and **downstream capture process**es. This query also identifies redo log files that are no longer needed by any capture process at the local database.

The query displays the following information for each registered archived redo log file:

- The capture process name of a capture process that uses the file
- The name and location of the file at the local site
- The lowest SCN value for the information contained in the redo log file
- The lowest SCN value for the next redo log file in the sequence
- Whether the redo log file is purgeable

To display this information about each registered archive redo log file in a database, run the following query:

```
COLUMN CONSUMER_NAME HEADING 'Capture|Process|Name' FORMAT A15
COLUMN NAME HEADING 'Archived Redo Log|File Name' FORMAT A25
COLUMN FIRST_SCN HEADING 'First SCN' FORMAT 99999999999
COLUMN NEXT_SCN HEADING 'Next SCN' FORMAT 99999999999
COLUMN PURGEABLE HEADING 'Purgeable?' FORMAT A10

SELECT r.CONSUMER_NAME,
       r.NAME,
       r.FIRST_SCN,
       r.NEXT_SCN,
       r.PURGEABLE
  FROM DBA_REGISTERED_ARCHIVED_LOG r, DBA_CAPTURE c
  WHERE r.CONSUMER_NAME = c.CAPTURE_NAME;
```

Your output looks similar to the following:

```
Capture
Process         Archived Redo Log
Name            File Name                     First SCN     Next SCN Purgeable?
--------------- ------------------------- ------------ ------------ ----------
CAPTURE_SIMP    /private1/ARCHIVE_LOGS/1_       509686       549100 YES
                3_502628294.dbf

CAPTURE_SIMP    /private1/ARCHIVE_LOGS/1_       549100       587296 YES
                4_502628294.dbf

CAPTURE_SIMP    /private1/ARCHIVE_LOGS/1_       587296       623107 NO
                5_502628294.dbf
```

The redo log files with `YES` for `Purgeable?` for all capture processes will never be needed by any capture process at the local database. These redo log files can be removed without affecting any existing capture process at the local database. The redo log files with `NO` for `Purgeable?` for one or more capture processes must be retained.

See Also: "ARCHIVELOG Mode and a Capture Process" on page 2-38

Displaying the Last Archived Redo Entry Available to Each Capture Process

For a **local capture process**, the last archived redo entry available is the last entry from the online redo log flushed to an archived log file. For a **downstream capture process**, the last archived redo entry available is the redo entry with the most recent SCN in the last archived log file added to the LogMiner session used by the capture process.

You can display the following information about the last redo entry that was made available to each capture process by running the query in this section:

- The name of the capture process
- The identification number of the LogMiner session used by the capture process
- The SCN of the last redo entry available for the capture process
- The time when the last redo entry became available for the capture process

The information displayed by this query is valid only for an enabled capture process.

Run the following query to display this information for each capture process:

```
COLUMN CAPTURE_NAME HEADING 'Capture|Name' FORMAT A20
COLUMN LOGMINER_ID HEADING 'LogMiner ID' FORMAT 9999
COLUMN AVAILABLE_MESSAGE_NUMBER HEADING 'Last Redo SCN' FORMAT 9999999999
COLUMN AVAILABLE_MESSAGE_CREATE_TIME HEADING 'Time of|Last Redo SCN'

SELECT CAPTURE_NAME,
       LOGMINER_ID,
       AVAILABLE_MESSAGE_NUMBER,
       TO_CHAR(AVAILABLE_MESSAGE_CREATE_TIME, 'HH24:MI:SS MM/DD/YY')
         AVAILABLE_MESSAGE_CREATE_TIME
  FROM V$STREAMS_CAPTURE;
```

Your output looks similar to the following:

```
Capture                                          Time of
Name                 LogMiner ID Last Redo SCN Last Redo SCN
-------------------- ----------- ------------- -----------------
STREAMS_CAPTURE                1        322953 11:33:20 10/16/03
```

Listing the Parameter Settings for Each Capture Process

The following query displays the current setting for each **capture process** parameter for each capture process in a database:

```
COLUMN CAPTURE_NAME HEADING 'Capture|Process|Name' FORMAT A25
COLUMN PARAMETER HEADING 'Parameter' FORMAT A25
COLUMN VALUE HEADING 'Value' FORMAT A10
COLUMN SET_BY_USER HEADING 'Set by User?' FORMAT A15

SELECT CAPTURE_NAME,
       PARAMETER,
       VALUE,
       SET_BY_USER
  FROM DBA_CAPTURE_PARAMETERS;
```

Your output looks similar to the following:

```
Capture
Process
Name                      Parameter                 Value      Set by User?
------------------------- ------------------------- ---------- ---------------
CAPTURE                   DISABLE_ON_LIMIT          N          NO
CAPTURE                   DOWNSTREAM_REAL_TIME_MINE Y          NO
CAPTURE                   MAXIMUM_SCN               INFINITE   NO
CAPTURE                   MESSAGE_LIMIT             INFINITE   NO
CAPTURE                   PARALLELISM               3          YES
CAPTURE                   STARTUP_SECONDS           0          NO
CAPTURE                   TIME_LIMIT                INFINITE   NO
CAPTURE                   TRACE_LEVEL               0          NO
CAPTURE                   WRITE_ALERT_LOG           Y          NO
```

Note: If the `Set by User?` column is `NO` for a parameter, then the parameter is set to its default value. If the `Set by User?` column is `YES` for a parameter, then the parameter might or might not be set to its default value.

See Also:

- "Capture Process Architecture" on page 2-22
- "Setting a Capture Process Parameter" on page 11-28

Viewing the Extra Attributes Captured by Each Capture Process

You can use the `INCLUDE_EXTRA_ATTRIBUTE` procedure in the `DBMS_CAPTURE_ADM` package to instruct a **capture process** to capture one or more extra attributes from the redo log. The following query displays the extra attributes included in the LCRs captured by each capture process in the local database:

```
COLUMN CAPTURE_NAME HEADING 'Capture Process' FORMAT A20
COLUMN ATTRIBUTE_NAME HEADING 'Attribute Name' FORMAT A15
COLUMN INCLUDE HEADING 'Include Attribute in LCRs?' FORMAT A30

SELECT CAPTURE_NAME, ATTRIBUTE_NAME, INCLUDE
  FROM DBA_CAPTURE_EXTRA_ATTRIBUTES
  ORDER BY CAPTURE_NAME;
```

Your output looks similar to the following:

```
Capture Process      Attribute Name  Include Attribute in LCRs?
-------------------- --------------- ------------------------------
STREAMS_CAPTURE      ROW_ID          NO
STREAMS_CAPTURE      SERIAL#         NO
STREAMS_CAPTURE      SESSION#        NO
STREAMS_CAPTURE      THREAD#         NO
STREAMS_CAPTURE      TX_NAME         YES
STREAMS_CAPTURE      USERNAME        NO
```

Based on this output, the capture process named `streams_capture` includes the transaction name (`tx_name`) in the LCRs that it captures, but this capture process does not include any other extra attributes in the LCRs that it captures.

See Also:

- "Extra Information in LCRs" on page 2-5
- "Managing Extra Attributes in Captured Messages" on page 11-33
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `INCLUDE_EXTRA_ATTRIBUTE` procedure

Determining the Applied SCN for All Capture Processes in a Database

The applied system change number (SCN) for a **capture process** is the SCN of the most recent **message** dequeued by the relevant **apply process**es. All changes below this **applied SCN** have been dequeued by all apply processes that apply changes captured by the capture process.

To display the applied SCN for all of the capture processes in a database, run the following query:

```
COLUMN CAPTURE_NAME HEADING 'Capture Process Name' FORMAT A30
COLUMN APPLIED_SCN HEADING 'Applied SCN' FORMAT 99999999999

SELECT CAPTURE_NAME, APPLIED_SCN FROM DBA_CAPTURE;
```

Your output looks similar to the following:

```
Capture Process Name           Applied SCN
------------------------------ -----------
CAPTURE_EMP                        177154
```

Determining Redo Log Scanning Latency for Each Capture Process

You can find the following information about each **capture process** by running the query in this section:

- The redo log scanning latency, which specifies the number of seconds between the creation time of the most recent redo log entry scanned by a capture process and the current time. This number might be relatively large immediately after you start a capture process.
- The seconds since last recorded status, which is the number of seconds since a capture process last recorded its status.
- The current capture process time, which is the latest time when the capture process recorded its status.
- The **message** creation time, which is the time when the data manipulation language (DML) or data definition language (DDL) change generated the redo data for the most recently **captured message**.

The information displayed by this query is valid only for an enabled capture process.

Run the following query to determine the redo scanning latency for each capture process:

```
COLUMN CAPTURE_NAME HEADING 'Capture|Process|Name' FORMAT A10
COLUMN LATENCY_SECONDS HEADING 'Latency|in|Seconds' FORMAT 999999
COLUMN LAST_STATUS HEADING 'Seconds Since|Last Status' FORMAT 999999
COLUMN CAPTURE_TIME HEADING 'Current|Process|Time'
COLUMN CREATE_TIME HEADING 'Message|Creation Time' FORMAT 999999

SELECT CAPTURE_NAME,
       ((SYSDATE - CAPTURE_MESSAGE_CREATE_TIME)*86400) LATENCY_SECONDS,
       ((SYSDATE - CAPTURE_TIME)*86400) LAST_STATUS,
       TO_CHAR(CAPTURE_TIME, 'HH24:MI:SS MM/DD/YY') CAPTURE_TIME,
       TO_CHAR(CAPTURE_MESSAGE_CREATE_TIME, 'HH24:MI:SS MM/DD/YY') CREATE_TIME
  FROM V$STREAMS_CAPTURE;
```

Your output looks similar to the following:

```
Capture    Latency               Current
Process         in Seconds Since Process           Message
Name       Seconds   Last Status Time              Creation Time
---------- ------- ------------- ----------------- -----------------
CAPTURE          4             4 12:04:13 03/01/02 12:04:13 03/01/02
```

The `"Latency in Seconds"` returned by this query is the difference between the current time (`SYSDATE`) and the `"Message Creation Time."` The `"Seconds Since Last Status"` returned by this query is the difference between the current time (`SYSDATE`) and the `"Current Process Time."`

Determining Message Enqueuing Latency for Each Capture Process

You can find the following information about each **capture process** by running the query in this section:

- The **message** enqueuing latency, which specifies the number of seconds between when an entry was recorded in the redo log and when the message was enqueued by the capture process
- The message creation time, which is the time when the data manipulation language (DML) or data definition language (DDL) change generated the redo data for the most recently enqueued message
- The enqueue time, which is when the capture process enqueued the message into its **queue**
- The message number of the enqueued message

The information displayed by this query is valid only for an enabled capture process.

Run the following query to determine the message capturing latency for each capture process:

```
COLUMN CAPTURE_NAME HEADING 'Capture|Process|Name' FORMAT A10
COLUMN LATENCY_SECONDS HEADING 'Latency|in|Seconds' FORMAT 999999
COLUMN CREATE_TIME HEADING 'Message Creation|Time' FORMAT A20
COLUMN ENQUEUE_TIME HEADING 'Enqueue Time' FORMAT A20
COLUMN ENQUEUE_MESSAGE_NUMBER HEADING 'Message|Number' FORMAT 999999

SELECT CAPTURE_NAME,
       (ENQUEUE_TIME-ENQUEUE_MESSAGE_CREATE_TIME)*86400 LATENCY_SECONDS,
       TO_CHAR(ENQUEUE_MESSAGE_CREATE_TIME, 'HH24:MI:SS MM/DD/YY') CREATE_TIME,
       TO_CHAR(ENQUEUE_TIME, 'HH24:MI:SS MM/DD/YY') ENQUEUE_TIME,
       ENQUEUE_MESSAGE_NUMBER
  FROM V$STREAMS_CAPTURE;
```

Your output looks similar to the following:

```
Capture    Latency
Process         in Message Creation                          Message
Name       Seconds Time                 Enqueue Time         Number
---------- ------- -------------------- -------------------- -------
CAPTURE          0 10:56:51 03/01/02    10:56:51 03/01/02     253962
```

The `"Latency in Seconds"` returned by this query is the difference between the `"Enqueue Time"` and the `"Message Creation Time."`

Displaying Information About Rule Evaluations for Each Capture Process

You can display the following information about **rule** evaluation for each **capture process** by running the query in this section:

- The name of the capture process.
- The number of changes discarded during prefiltering since the capture process was last started. The capture process determined that these changes definitely did not satisfy the capture process **rule set**s during prefiltering.
- The number of changes kept during prefiltering since the capture process was last started. The capture process determined that these changes definitely satisfied the capture process rule sets during prefiltering. Such changes are converted into LCRs and enqueued into the capture process **queue**.

- The total number of prefilter evaluations since the capture process was last started.
- The number of undecided changes after prefiltering since the capture process was last started. These changes might or might not satisfy the capture process rule sets. Some of these changes might be filtered out after prefiltering without requiring full evaluation. Other changes require full evaluation to determine whether they satisfy the capture process rule sets.
- The number of full evaluations since the capture process was last started. Full evaluations can be expensive. Therefore, capture process performance is best when this number is relatively low.

The information displayed by this query is valid only for an enabled capture process.

Run the following query to display this information for each capture process:

```
COLUMN CAPTURE_NAME HEADING 'Capture|Name' FORMAT A15
COLUMN TOTAL_PREFILTER_DISCARDED HEADING 'Prefilter|Changes|Discarded'
  FORMAT 9999999999
COLUMN TOTAL_PREFILTER_KEPT HEADING 'Prefilter|Changes|Kept' FORMAT 9999999999
COLUMN TOTAL_PREFILTER_EVALUATIONS HEADING 'Prefilter|Evaluations'
  FORMAT 9999999999
COLUMN UNDECIDED HEADING 'Undecided|After|Prefilter' FORMAT 9999999999
COLUMN TOTAL_FULL_EVALUATIONS HEADING 'Full|Evaluations' FORMAT 9999999999

SELECT CAPTURE_NAME,
       TOTAL_PREFILTER_DISCARDED,
       TOTAL_PREFILTER_KEPT,
       TOTAL_PREFILTER_EVALUATIONS,
       (TOTAL_PREFILTER_EVALUATIONS -
         (TOTAL_PREFILTER_KEPT + TOTAL_PREFILTER_DISCARDED)) UNDECIDED,
       TOTAL_FULL_EVALUATIONS
  FROM V$STREAMS_CAPTURE;
```

Your output looks similar to the following:

```
Capture     Prefilter   Prefilter               Undecided
Process       Changes     Changes   Prefilter       After        Full
Name        Discarded        Kept Evaluations   Prefilter Evaluations
---------- ---------- ----------- ----------- ----------- -----------
STM1$CAP       219493           2      219961         466           0
```

The total number of prefilter evaluations equals the sum of the prefilter changes discarded, the prefilter changes kept, and the undecided changes.

See Also: "Capture Process Rule Evaluation" on page 2-41

21

Monitoring Streams Queues and Propagations

This chapter provides sample queries that you can use to monitor Streams **queue**s and **propagation**s.

This chapter contains these topics:

- Monitoring ANYDATA Queues and Messaging
- Monitoring Buffered Queues
- Monitoring Streams Propagations and Propagation Jobs

Note: The Streams tool in the Oracle Enterprise Manager Console is also an excellent way to monitor a Streams environment. See the online help for the Streams tool for more information.

See Also:

- Chapter 3, "Streams Staging and Propagation"
- Chapter 12, "Managing Staging and Propagation"
- *Oracle Database Reference* for information about the data dictionary views described in this chapter
- *Oracle Streams Replication Administrator's Guide* for information about monitoring a Streams **replication** environment

Monitoring ANYDATA Queues and Messaging

The following sections contain instructions for displaying information about `ANYDATA` queues and messaging:

- Displaying the ANYDATA Queues in a Database
- Viewing the Messaging Clients in a Database
- Viewing Message Notifications
- Determining the Consumer of Each User-Enqueued Message in a Queue
- Viewing the Contents of User-Enqueued Messages in a Queue

See Also:

- Chapter 3, "Streams Staging and Propagation"
- Chapter 12, "Managing Staging and Propagation"

Displaying the ANYDATA Queues in a Database

To display all of the ANYDATA queues in a database, run the following query:

```
COLUMN OWNER HEADING 'Owner' FORMAT A10
COLUMN NAME HEADING 'Queue Name' FORMAT A28
COLUMN QUEUE_TABLE HEADING 'Queue Table' FORMAT A22
COLUMN USER_COMMENT HEADING 'Comment' FORMAT A15

SELECT q.OWNER, q.NAME, t.QUEUE_TABLE, q.USER_COMMENT
  FROM DBA_QUEUES q, DBA_QUEUE_TABLES t
  WHERE t.OBJECT_TYPE = 'SYS.ANYDATA' AND
        q.QUEUE_TABLE = t.QUEUE_TABLE AND
        q.OWNER       = t.OWNER;
```

Your output looks similar to the following:

```
Owner      Queue Name                    Queue Table            Comment
---------- ----------------------------- ---------------------- ---------------
SYS        AQ$_SCHEDULER$_JOBQTAB_E      SCHEDULER$_JOBQTAB     exception queue
SYS        SCHEDULER$_JOBQ               SCHEDULER$_JOBQTAB     Scheduler job q
                                                                ueue
SYS        AQ$_DIR$EVENT_TABLE_E         DIR$EVENT_TABLE        exception queue
SYS        DIR$EVENT_QUEUE               DIR$EVENT_TABLE
SYS        AQ$_DIR$CLUSTER_DIR_TABLE_E   DIR$CLUSTER_DIR_TABLE  exception queue
SYS        DIR$CLUSTER_DIR_QUEUE         DIR$CLUSTER_DIR_TABLE
STRMADMIN  AQ$_STREAMS_QUEUE_TABLE_E     STREAMS_QUEUE_TABLE    exception queue
STRMADMIN  STREAMS_QUEUE                 STREAMS_QUEUE_TABLE
```

An **exception queue** is created automatically when you create an ANYDATA queue.

See Also: "Managing ANYDATA Queues" on page 12-1

Viewing the Messaging Clients in a Database

You can view the **messaging client**s in a database by querying the DBA_STREAMS_MESSAGE_CONSUMERS data dictionary view. The query in this section displays the following information about each messaging client:

- The name of the messaging client
- The **queue** used by the messaging client
- The **positive rule set** used by the messaging client
- The **negative rule set** used by the messaging client

Run the following query to view this information about messaging clients:

```
COLUMN STREAMS_NAME HEADING 'Messaging|Client' FORMAT A25
COLUMN QUEUE_OWNER HEADING 'Queue|Owner' FORMAT A10
COLUMN QUEUE_NAME HEADING 'Queue Name' FORMAT A18
COLUMN RULE_SET_NAME HEADING 'Positive|Rule Set' FORMAT A11
COLUMN NEGATIVE_RULE_SET_NAME HEADING 'Negative|Rule Set' FORMAT A11
```

```
SELECT STREAMS_NAME,
       QUEUE_OWNER,
       QUEUE_NAME,
       RULE_SET_NAME,
       NEGATIVE_RULE_SET_NAME
  FROM DBA_STREAMS_MESSAGE_CONSUMERS;
```

Your output looks similar to the following:

```
Messaging                Queue                          Positive    Negative
Client                   Owner      Queue Name          Rule Set    Rule Set
------------------------ ---------- ------------------ ----------- -----------
SCHEDULER_PICKUP         SYS        SCHEDULER$_JOBQ    RULESET$_8
SCHEDULER_COORDINATOR    SYS        SCHEDULER$_JOBQ    RULESET$_4
HR                       STRMADMIN  STREAMS_QUEUE      RULESET$_15
```

See Also: Chapter 3, "Streams Staging and Propagation" for more information about messaging clients

Viewing Message Notifications

You can configure a **message** notification to send a notification when a message that can be dequeued by a **messaging client** is enqueued into a **queue**. The notification can be sent to an email address, to an HTTP URL, or to a PL/SQL procedure. Run the following query to view the message notifications configured in a database:

```
COLUMN STREAMS_NAME HEADING 'Messaging|Client' FORMAT A10
COLUMN QUEUE_OWNER HEADING 'Queue|Owner' FORMAT A5
COLUMN QUEUE_NAME HEADING 'Queue Name' FORMAT A20
COLUMN NOTIFICATION_TYPE HEADING 'Notification|Type' FORMAT A15
COLUMN NOTIFICATION_ACTION HEADING 'Notification|Action' FORMAT A25

SELECT STREAMS_NAME,
       QUEUE_OWNER,
       QUEUE_NAME,
       NOTIFICATION_TYPE,
       NOTIFICATION_ACTION
  FROM DBA_STREAMS_MESSAGE_CONSUMERS
  WHERE NOTIFICATION_TYPE IS NOT NULL;
```

Your output looks similar to the following:

```
Messaging  Queue                      Notification    Notification
Client     Owner Queue Name           Type            Action
---------- ----- -------------------- --------------- -------------------------
OE         OE    NOTIFICATION_QUEUE   MAIL            mary.smith@mycompany.com
```

See Also: "Configuring a Messaging Client and Message Notification" on page 12-18

Determining the Consumer of Each User-Enqueued Message in a Queue

To determine the consumer for each **user-enqueued message** in a **queue**, query `AQ$`*`queue_table_name`* in the queue owner's schema, where *`queue_table_name`* is the name of the **queue table**. For example, to find the consumers of the user-enqueued messages in the `oe_q_table_any` queue table, run the following query:

```
COLUMN MSG_ID HEADING 'Message ID' FORMAT 9999
COLUMN MSG_STATE HEADING 'Message State' FORMAT A13
COLUMN CONSUMER_NAME HEADING 'Consumer' FORMAT A30

SELECT MSG_ID, MSG_STATE, CONSUMER_NAME FROM AQ$OE_Q_TABLE_ANY;
```

Your output looks similar to the following:

```
Message ID                       Message State Consumer
-------------------------------- ------------- ------------------------------
B79AC412AE6E08CAE034080020AE3E0A PROCESSED     OE
B79AC412AE6F08CAE034080020AE3E0A PROCESSED     OE
B79AC412AE7008CAE034080020AE3E0A PROCESSED     OE
```

Note: This query lists only user-enqueued messages, not **captured messages**.

See Also: *Oracle Streams Advanced Queuing User's Guide and Reference* for an example that enqueues messages into an ANYDATA queue

Viewing the Contents of User-Enqueued Messages in a Queue

In an ANYDATA queue, to view the contents of a payload that is encapsulated within an ANYDATA payload, you query the **queue table** using the Access*data_type* static functions of the ANYDATA type, where *data_type* is the type of payload to view.

See Also: "Wrapping User Message Payloads in an ANYDATA Wrapper and Enqueuing Them" on page 12-15 for an example that enqueues the **message**s shown in the queries in this section into an ANYDATA queue

For example, to view the contents of payload of type NUMBER in a **queue** with a queue table named oe_queue_table, run the following query as the queue owner:

```
SELECT qt.user_data.AccessNumber() "Numbers in Queue"
  FROM strmadmin.oe_q_table_any qt;
```

Your output looks similar to the following:

```
Numbers in Queue
----------------
              16
```

Similarly, to view the contents of a payload of type VARCHAR2 in a queue with a queue table named oe_q_table_any, run the following query:

```
SELECT qt.user_data.AccessVarchar2() "Varchar2s in Queue"
   FROM strmadmin.oe_q_table_any qt;
```

Your output looks similar to the following:

```
Varchar2s in Queue
--------------------------------------------------------------------------------
Chemicals - SW
```

To view the contents of a user-defined datatype, you query the queue table using a custom function that you create. For example, to view the contents of a payload of

oe.cust_address_typ, connect as the Streams administrator and create a function similar to the following:

```
CONNECT oe/oe

CREATE OR REPLACE FUNCTION oe.view_cust_address_typ(
in_any IN ANYDATA)
RETURN oe.cust_address_typ
IS
  address   oe.cust_address_typ;
  num_var   NUMBER;
BEGIN
  IF (in_any.GetTypeName() = 'OE.CUST_ADDRESS_TYP') THEN
    num_var := in_any.GetObject(address);
    RETURN address;
  ELSE RETURN NULL;
  END IF;
END;
/

GRANT EXECUTE ON oe.view_cust_address_typ TO strmadmin;

GRANT EXECUTE ON oe.cust_address_typ TO strmadmin;
```

Query the queue table using the function, as in the following example:

```
CONNECT strmadmin/strmadminpw

SELECT oe.view_cust_address_typ(qt.user_data) "Customer Addresses"
  FROM strmadmin.oe_q_table_any qt
  WHERE qt.user_data.GetTypeName() = 'OE.CUST_ADDRESS_TYP';
```

Your output looks similar to the following:

```
Customer Addresses(STREET_ADDRESS, POSTAL_CODE, CITY, STATE_PROVINCE, COUNTRY_ID
--------------------------------------------------------------------------------
CUST_ADDRESS_TYP('1646 Brazil Blvd', '361168', 'Chennai', 'Tam', 'IN')
```

Monitoring Buffered Queues

A **buffered queue** includes the following storage areas:

- System Global Area (SGA) memory associated with a **queue**
- Part of the **queue table** for a queue that stores **message**s that have spilled from memory

Buffered queues are stored in the **Streams pool**, and the Streams pool is a portion of memory in the System Global Area (SGA) that is used by Streams. In a Streams environment, LCRs captured by a **capture process** always are stored in the buffered queue of an ANYDATA queue. Users and application can also enqueue messages into buffered queues, and these buffered queues be part of ANYDATA queues or part of **typed queue**s.

Buffered queues enable Oracle databases to optimize messages by storing them in the SGA instead of always storing them in a queue table. Captured messages always are stored in buffered queues, but user-enqueued LCRs and **user message**s can be stored in buffered queues or persistently in queue tables. Messages in a buffered queue can spill from memory if they have been staged in the buffered queue for a period of time without being dequeued, or if there is not enough space in memory to hold all of the messages. Messages that spill from memory are stored in the appropriate queue table.

The following sections describe queries that monitor buffered queues:

- Determining the Number of Messages in Each Buffered Queue
- Viewing the Capture Processes for the LCRs in Each Buffered Queue
- Displaying Information About Propagations that Send Buffered Messages
- Displaying the Number of Messages and Bytes Sent By Propagations
- Displaying Performance Statistics for Propagations that Send Buffered Messages
- Viewing the Propagations Dequeuing Messages from Each Buffered Queue
- Displaying Performance Statistics for Propagations that Receive Buffered Messages
- Viewing the Apply Processes Dequeuing Messages from Each Buffered Queue

Determining the Number of Messages in Each Buffered Queue

The `V$BUFFERED_QUEUES` dynamic performance view contains information about the number of **message**s in a **buffered queue**. The messages can be **captured message**s, or **user-enqueued message**s, or both.

You can determine the following information about each buffered queue in a database by running the query in this section:

- The **queue** owner
- The queue name
- The number of messages currently in memory
- The number of messages that have spilled from memory into the **queue table**
- The total number of messages in the buffered queue, which includes the messages in memory and the messages spilled to the queue table

To display this information, run the following query:

```
COLUMN QUEUE_SCHEMA HEADING 'Queue Owner' FORMAT A15
COLUMN QUEUE_NAME HEADING 'Queue Name' FORMAT A15
COLUMN MEM_MSG HEADING 'Messages|in Memory' FORMAT 99999999
COLUMN SPILL_MSGS HEADING 'Messages|Spilled' FORMAT 99999999
COLUMN NUM_MSGS HEADING 'Total Messages|in Buffered Queue' FORMAT 99999999

SELECT QUEUE_SCHEMA,
       QUEUE_NAME,
       (NUM_MSGS - SPILL_MSGS) MEM_MSG,
       SPILL_MSGS,
       NUM_MSGS
  FROM V$BUFFERED_QUEUES;
```

Your output looks similar to the following:

```
                                    Messages      Messages      Total Messages
Queue Owner     Queue Name          in Memory      Spilled   in Buffered Queue
--------------- --------------- ------------ ------------ ------------------
STRMADMIN       STREAMS_QUEUE            534           21                555
```

Viewing the Capture Processes for the LCRs in Each Buffered Queue

A **capture process** is a **queue** publisher that enqueues **captured message**s into a **buffered queue**. These LCRs can be propagated to other queues subsequently. By querying the V$BUFFERED_PUBLISHERS dynamic performance view, you can display each capture process that captured the LCRs in the buffered queue. These LCRs might have been captured at the local database, or they might have been captured at a remote database and propagated to the queue specified in the query.

The query in this section assumes that the buffered queues in the local database only store captured messages, not **user-enqueued message**s. The query displays the following information about each capture process:

- The name of a capture process that captured the LCRs in the buffered queue
- If the capture process is running on a remote database, and the captured messages have been propagated to the local queue, then the name of the queue and database from which the captured messages were last propagated
- The name of the local queue staging the captured messages
- The total number of LCRs captured by a capture process that have been staged in the buffered queue since the database instance was last started
- The message number of the LCR last enqueued into the buffered queue from the sender

To display this information, run the following query:

```
COLUMN SENDER_NAME HEADING 'Capture|Process' FORMAT A13
COLUMN SENDER_ADDRESS HEADING 'Sender Queue' FORMAT A27
COLUMN QUEUE_NAME HEADING 'Queue Name' FORMAT A15
COLUMN CNUM_MSGS HEADING 'Number|of LCRs|Enqueued' FORMAT 99999999
COLUMN LAST_ENQUEUED_MSG HEADING 'Last|Enqueued|LCR' FORMAT 99999999

SELECT SENDER_NAME,
       SENDER_ADDRESS,
       QUEUE_NAME,
       CNUM_MSGS,
       LAST_ENQUEUED_MSG
  FROM V$BUFFERED_PUBLISHERS;
```

Your output looks similar to the following:

```
                                                          Number      Last
Capture                                                   of LCRs  Enqueued
Process       Sender Queue                Queue Name      Enqueued       LCR
------------- --------------------------- --------------- --------- ---------
CAPTURE_HR    "STRMADMIN"."STREAMS_QUEUE" STREAMS_QUEUE         382       844
              @MULT3.NET

CAPTURE_HR    "STRMADMIN"."STREAMS_QUEUE" STREAMS_QUEUE         387       840
              @MULT2.NET

CAPTURE_HR                                STREAMS_QUEUE          75       833
```

This output shows following:

- 382 LCRs from the capture_hr capture process running on a remote database were propagated from a queue named streams_queue on database mult3.net to the local queue named streams_queue. The message number of the last enqueued LCR from this sender was 844.
- 387 LCRs from the capture_hr capture process running on a remote database were propagated from a queue named streams_queue on database mult2.net to the local queue named streams_queue. The message number of the last enqueued LCR from this sender was 840.
- 75 LCRs from the local capture_hr capture process were enqueued into the local queue named streams_queue. The capture process is local because the Sender Queue column is NULL. The message number of the last enqueued LCR from this capture process was 833.

Displaying Information About Propagations that Send Buffered Messages

The query in this section displays the following information about each **propagation** that sends buffered **message**s from a **buffered queue** in the local database:

- The name of the propagation
- The **queue** owner
- The queue name
- The name of the database link used by the propagation
- The status of the **propagation schedule**

To display this information, run the following query:

```
COLUMN PROPAGATION_NAME HEADING 'Propagation' FORMAT A15
COLUMN QUEUE_SCHEMA HEADING 'Queue|Owner' FORMAT A10
COLUMN QUEUE_NAME HEADING 'Queue|Name' FORMAT A15
COLUMN DBLINK HEADING 'Database|Link' FORMAT A10
COLUMN SCHEDULE_STATUS HEADING 'Schedule Status' FORMAT A20

SELECT p.PROPAGATION_NAME,
       s.QUEUE_SCHEMA,
       s.QUEUE_NAME,
       s.DBLINK,
       s.SCHEDULE_STATUS
  FROM DBA_PROPAGATION p, V$PROPAGATION_SENDER s
  WHERE p.DESTINATION_DBLINK = s.DBLINK AND
        p.SOURCE_QUEUE_OWNER = s.QUEUE_SCHEMA AND
        p.SOURCE_QUEUE_NAME  = s.QUEUE_NAME;
```

Your output looks similar to the following:

```
                Queue      Queue           Database
Propagation     Owner      Name            Link       Schedule Status
--------------- ---------- --------------- ---------- --------------------
MULT1_TO_MULT3  STRMADMIN  STREAMS_QUEUE   MULT3.NET  SCHEDULE ENABLED
MULT1_TO_MULT2  STRMADMIN  STREAMS_QUEUE   MULT2.NET  SCHEDULE ENABLED
```

Displaying the Number of Messages and Bytes Sent By Propagations

The query in this section displays the number of **message**s and the number of bytes sent by each **propagation** that sends buffered messages from a **buffered queue** in the local database:

- The name of the propagation
- The **queue** name
- The name of the database link used by the propagation
- The total number of messages sent since the database instance was last started
- The total number of bytes sent since the database instance was last started

To display this information, run the following query:

```
COLUMN PROPAGATION_NAME HEADING 'Propagation' FORMAT A15
COLUMN QUEUE_NAME HEADING 'Queue|Name' FORMAT A15
COLUMN DBLINK HEADING 'Database|Link' FORMAT A10
COLUMN TOTAL_MSGS HEADING 'Total|Messages' FORMAT 99999999
COLUMN TOTAL_BYTES HEADING 'Total|Bytes' FORMAT 99999999

SELECT p.PROPAGATION_NAME,
       s.QUEUE_NAME,
       s.DBLINK,
       s.TOTAL_MSGS,
       s.TOTAL_BYTES
  FROM DBA_PROPAGATION p, V$PROPAGATION_SENDER s
  WHERE p.DESTINATION_DBLINK = s.DBLINK AND
        p.SOURCE_QUEUE_OWNER = s.QUEUE_SCHEMA AND
        p.SOURCE_QUEUE_NAME  = s.QUEUE_NAME;
```

Your output looks similar to the following:

```
                Queue           Database       Total     Total
Propagation     Name            Link        Messages     Bytes
--------------- --------------- ---------- --------- ---------
MULT1_TO_MULT3  STREAMS_QUEUE   MULT3.NET         79     71467
MULT1_TO_MULT2  STREAMS_QUEUE   MULT2.NET         79     71467
```

Displaying Performance Statistics for Propagations that Send Buffered Messages

The query in this section displays the amount of time that a **propagation** sending buffered **message**s spends performing various tasks. Each propagation sends messages from the **source queue** to the **destination queue**. Specifically, the query displays the following information:

- The name of the propagation
- The **queue** name
- The name of the database link used by the propagation
- The amount of time spent dequeuing messages from the queue since the database instance was last started, in seconds
- The amount of time spent pickling messages since the database instance was last started, in seconds. Pickling involves changing a message in memory into a series of bytes that can be sent over a network.
- The amount of time spent propagating messages since the database instance was last started, in seconds

To display this information, run the following query:

```
COLUMN PROPAGATION_NAME HEADING 'Propagation' FORMAT A15
COLUMN QUEUE_NAME HEADING 'Queue|Name' FORMAT A13
COLUMN DBLINK HEADING 'Database|Link' FORMAT A9
COLUMN ELAPSED_DEQUEUE_TIME HEADING 'Dequeue|Time' FORMAT 99999999.99
COLUMN ELAPSED_PICKLE_TIME HEADING 'Pickle|Time' FORMAT 99999999.99
COLUMN ELAPSED_PROPAGATION_TIME HEADING 'Propagation|Time' FORMAT 99999999.99

SELECT p.PROPAGATION_NAME,
       s.QUEUE_NAME,
       s.DBLINK,
       (s.ELAPSED_DEQUEUE_TIME / 100) ELAPSED_DEQUEUE_TIME,
       (s.ELAPSED_PICKLE_TIME / 100) ELAPSED_PICKLE_TIME,
       (s.ELAPSED_PROPAGATION_TIME / 100) ELAPSED_PROPAGATION_TIME
  FROM DBA_PROPAGATION p, V$PROPAGATION_SENDER s
  WHERE p.DESTINATION_DBLINK = s.DBLINK AND
        p.SOURCE_QUEUE_OWNER = s.QUEUE_SCHEMA AND
        p.SOURCE_QUEUE_NAME  = s.QUEUE_NAME;
```

Your output looks similar to the following:

```
                Queue         Database       Dequeue       Pickle  Propagation
Propagation     Name          Link             Time         Time         Time
--------------- ------------- --------- ------------ ------------ ------------
MULT1_TO_MULT2  STREAMS_QUEUE MULT2.NET        30.65        45.10        10.91
MULT1_TO_MULT3  STREAMS_QUEUE MULT3.NET        25.36        37.07         8.35
```

Viewing the Propagations Dequeuing Messages from Each Buffered Queue

Propagations are **queue** subscribers that can dequeue **message**s from a queue. By querying the `V$BUFFERED_SUBSCRIBERS` dynamic performance view, you can display all the **propagation**s that can dequeue buffered messages from a queue.

You can also use the `V$BUFFERED_SUBSCRIBERS` dynamic performance view to determine the performance of a propagation. For example, if a propagation has a high number of spilled messages, then that propagation might not be dequeuing messages fast enough from the **buffered queue**. Spilling messages to a **queue table** has a negative impact on the performance of your Streams environment.

Apply processes also are queue subscribers. This query joins with the `DBA_PROPAGATION` and `V$BUFFERED_QUEUES` views to limit the output to propagations only and to show the propagation name of each propagation.

The query in this section displays the following information about each propagation that can dequeue messages from queues:

- The name of the propagation.
- The **destination database**, which is the database that contains the **destination queue** for the propagation.
- The sequence number for the message most recently enqueued into the queue. The sequence number for message shows the order of the message in the queue.
- The sequence number for the message in the queue most recently browsed by the propagation.
- The sequence number for the message most recently dequeued from the queue by the propagation.

- The current number of messages in the queue waiting to be dequeued by the propagation.
- The cumulative number of messages spilled from memory to the queue table for the propagation since the database last started.

To display this information, run the following query:

```
COLUMN PROPAGATION_NAME HEADING 'Propagation' FORMAT A15
COLUMN SUBSCRIBER_ADDRESS HEADING 'Destination|Database' FORMAT A11
COLUMN CURRENT_ENQ_SEQ HEADING 'Current|Enqueued|Sequence' FORMAT 99999999
COLUMN LAST_BROWSED_SEQ HEADING 'Last|Browsed|Sequence' FORMAT 99999999
COLUMN LAST_DEQUEUED_SEQ HEADING 'Last|Dequeued|Sequence' FORMAT 99999999
COLUMN NUM_MSGS HEADING 'Number of|Messages|in Queue|(Current)' FORMAT 99999999
COLUMN TOTAL_SPILLED_MSG HEADING 'Number of|Spilled|Messages|(Cumulative)'
  FORMAT 99999999

SELECT p.PROPAGATION_NAME,
       s.SUBSCRIBER_ADDRESS,
       s.CURRENT_ENQ_SEQ,
       s.LAST_BROWSED_SEQ,
       s.LAST_DEQUEUED_SEQ,
       s.NUM_MSGS,
       s.TOTAL_SPILLED_MSG
FROM DBA_PROPAGATION p, V$BUFFERED_SUBSCRIBERS s, V$BUFFERED_QUEUES q
WHERE q.QUEUE_ID = s.QUEUE_ID AND
      p.SOURCE_QUEUE_OWNER = q.QUEUE_SCHEMA AND
      p.SOURCE_QUEUE_NAME = q.QUEUE_NAME AND
      p.DESTINATION_DBLINK = s.SUBSCRIBER_ADDRESS;
```

Your output looks similar to the following:

```
                                                                 Number of    Number of
                            Current      Last      Last  Messages      Spilled
                Destination Enqueued   Browsed  Dequeued in Queue     Messages
Propagation     Database    Sequence  Sequence  Sequence (Current) (Cumulative)
--------------- ----------- --------- --------- --------- --------- ------------
MULT1_TO_MULT2  MULT2.NET         157       144       129        24            0
MULT1_TO_MULT3  MULT3.NET          98        88        81        53            0
```

Note: If there are multiple propagations using the same database link but propagating messages to different queues at the destination database, then the statistics returned by this query are approximate rather than accurate.

Displaying Performance Statistics for Propagations that Receive Buffered Messages

The query in this section displays the amount of time that each **propagation** receiving buffered **message**s spends performing various tasks. Each propagation receives the messages and enqueues them into the **destination queue** for the propagation. Specifically, the query displays the following information:

- The name of the **source queue** from which messages are propagated.
- The name of the **source database**.
- The amount of time spent unpickling messages since the database instance was last started, in seconds. Unpickling involves changing a series of bytes that can be sent over a network back into a buffered message in memory.

- The amount of time spent evaluating **rule**s for propagated messages since the database instance was last started, in seconds.
- The amount of time spent enqueuing messages into the destination queue for the propagation since the database instance was last started, in seconds.

To display this information, run the following query:

```
COLUMN SRC_QUEUE_NAME HEADING 'Source|Queue|Name' FORMAT A20
COLUMN SRC_DBNAME HEADING 'Source|Database' FORMAT A15
COLUMN ELAPSED_UNPICKLE_TIME HEADING 'Unpickle|Time' FORMAT 99999999.99
COLUMN ELAPSED_RULE_TIME HEADING 'Rule|Evaluation|Time' FORMAT 99999999.99
COLUMN ELAPSED_ENQUEUE_TIME HEADING 'Enqueue|Time' FORMAT 99999999.99

SELECT SRC_QUEUE_NAME,
       SRC_DBNAME,
       (ELAPSED_UNPICKLE_TIME / 100) ELAPSED_UNPICKLE_TIME,
       (ELAPSED_RULE_TIME / 100) ELAPSED_RULE_TIME,
       (ELAPSED_ENQUEUE_TIME / 100) ELAPSED_ENQUEUE_TIME
  FROM V$PROPAGATION_RECEIVER;
```

Your output looks similar to the following:

```
Source                                                    Rule
Queue                Source             Unpickle   Evaluation      Enqueue
Name                 Database               Time         Time         Time
-------------------- -------------- ------------ ------------ ------------
STREAMS_QUEUE        MULT2.NET             45.65         5.44        45.85
STREAMS_QUEUE        MULT3.NET             53.35         8.01        50.41
```

Viewing the Apply Processes Dequeuing Messages from Each Buffered Queue

Apply processes are **queue** subscribers that can dequeue **message**s from a queue. By querying the `V$BUFFERED_SUBSCRIBERS` dynamic performance view, you can display all the apply processes that can dequeue messages from a queue.

You can also use the `V$BUFFERED_SUBSCRIBERS` dynamic performance view to determine the performance of an apply process. For example, if an apply process has a high number of spilled messages, then that apply process might not be dequeuing messages fast enough from the **buffered queue**. Spilling messages to a **queue table** has a negative impact on the performance of your Streams environment.

This query joins with the `V$BUFFERED_QUEUES` views to show the name of the queue. In addition, **propagation**s also are queue subscribers, and this query limits the output to subscribers where the `SUBSCRIBER_ADDRESS` is `NULL` to return only apply processes.

The query in this section displays the following information about the apply processes that can dequeue messages from queues:

- The name of the apply process.
- The queue owner.
- The queue name.
- The sequence number for the message most recently dequeued by the apply process. The sequence number for message shows the order of the message in the queue.

- The current number of messages in the queue waiting to be dequeued by the apply process.
- The cumulative number of messages spilled from memory to the queue table for the apply process since the database last started.

To display this information, run the following query:

```
COLUMN SUBSCRIBER_NAME HEADING 'Apply Process' FORMAT A16
COLUMN QUEUE_SCHEMA HEADING 'Queue|Owner' FORMAT A10
COLUMN QUEUE_NAME HEADING 'Queue|Name' FORMAT A15
COLUMN LAST_DEQUEUED_SEQ HEADING 'Last|Dequeued|Sequence' FORMAT 99999999
COLUMN NUM_MSGS HEADING 'Number of|Messages|in Queue|(Current)' FORMAT 99999999
COLUMN TOTAL_SPILLED_MSG HEADING 'Number of|Spilled|Messages|(Cumulative)'
  FORMAT 99999999

SELECT s.SUBSCRIBER_NAME,
       q.QUEUE_SCHEMA,
       q.QUEUE_NAME,
       s.LAST_DEQUEUED_SEQ,
       s.NUM_MSGS,
       s.TOTAL_SPILLED_MSG
FROM V$BUFFERED_QUEUES q, V$BUFFERED_SUBSCRIBERS s, DBA_APPLY a
WHERE q.QUEUE_ID = s.QUEUE_ID AND
      s.SUBSCRIBER_ADDRESS IS NULL AND
      s.SUBSCRIBER_NAME = a.APPLY_NAME;
```

Your output looks similar to the following:

```
                                                 Last Number of   Number of
                 Queue      Queue            Dequeued  Messages     Spilled
Apply Process    Owner      Name             Sequence  in Queue    Messages
                                                      (Current)(Cumulative)
---------------- ---------- --------------- --------- --------- ------------
APPLY_FROM_MULT3 STRMADMIN  STREAMS_QUEUE          49       148            0
APPLY_FROM_MULT2 STRMADMIN  STREAMS_QUEUE          85       241            1
```

Monitoring Streams Propagations and Propagation Jobs

The following sections contain queries that you can run to display information about **propagation**s and **propagation job**s:

- Displaying the Queues and Database Link for Each Propagation
- Determining the Source Queue and Destination Queue for Each Propagation
- Determining the Rule Sets for Each Propagation
- Displaying the Schedule for a Propagation Job
- Determining the Total Number of Messages and Bytes Propagated

See Also:

- Chapter 3, "Streams Staging and Propagation"
- "Managing Streams Propagations and Propagation Jobs" on page 12-6

Displaying the Queues and Database Link for Each Propagation

You can display information about each **propagation** by querying the DBA_PROPAGATION data dictionary view. This view contains information about each propagation with a **source queue** is at the local database.

The query in this section displays the following information about each propagation:

- The propagation name
- The source **queue** name
- The database link used by the propagation
- The **destination queue** name
- The status of the propagation, either ENABLED, DISABLED, or ABORTED
- Whether the propagation is a queue-to-queue propagation

To display this information about each propagation in a database, run the following query:

```
COLUMN PROPAGATION_NAME        HEADING 'Propagation|Name'   FORMAT A19
COLUMN SOURCE_QUEUE_NAME       HEADING 'Source|Queue|Name'  FORMAT A17
COLUMN DESTINATION_DBLINK      HEADING 'Database|Link'      FORMAT A9
COLUMN DESTINATION_QUEUE_NAME  HEADING 'Dest|Queue|Name'    FORMAT A15
COLUMN STATUS                  HEADING 'Status'             FORMAT A8
COLUMN QUEUE_TO_QUEUE          HEADING 'Queue-|to-|Queue?'  FORMAT A6

SELECT PROPAGATION_NAME,
       SOURCE_QUEUE_NAME,
       DESTINATION_DBLINK,
       DESTINATION_QUEUE_NAME,
       STATUS,
       QUEUE_TO_QUEUE
  FROM DBA_PROPAGATION;
```

Your output looks similar to the following:

```
                    Source                        Dest                      Queue-
Propagation         Queue             Database    Queue                     to-
Name                Name              Link        Name            Status    Queue?
------------------- ----------------- ----------- --------------- --------- ------
STREAMS_PROPAGATION STREAMS_CAPTURE_Q INST2.NET   STREAMS_APPLY_Q ENABLED   FALSE
```

Determining the Source Queue and Destination Queue for Each Propagation

You can determine the **source queue** and **destination queue** for each **propagation** by querying the DBA_PROPAGATION data dictionary view.

The query in this section displays the following information about each propagation:

- The propagation name
- The source **queue** owner
- The source queue name
- The database that contains the source queue
- The destination queue owner
- The destination queue name
- The database that contains the destination queue

To display this information about each propagation in a database, run the following query:

```
COLUMN PROPAGATION_NAME HEADING 'Propagation|Name' FORMAT A20
COLUMN SOURCE_QUEUE_OWNER HEADING 'Source|Queue|Owner' FORMAT A10
COLUMN 'Source Queue' HEADING 'Source|Queue' FORMAT A15
COLUMN DESTINATION_QUEUE_OWNER HEADING 'Dest|Queue|Owner'   FORMAT A10
COLUMN 'Destination Queue' HEADING 'Destination|Queue' FORMAT A15

SELECT p.PROPAGATION_NAME,
       p.SOURCE_QUEUE_OWNER,
       p.SOURCE_QUEUE_NAME ||'@'||
       g.GLOBAL_NAME "Source Queue",
       p.DESTINATION_QUEUE_OWNER,
       p.DESTINATION_QUEUE_NAME ||'@'||
       p.DESTINATION_DBLINK "Destination Queue"
  FROM DBA_PROPAGATION p, GLOBAL_NAME g;
```

Your output looks similar to the following:

```
                     Source                         Dest
Propagation          Queue      Source              Queue      Destination
Name                 Owner      Queue               Owner      Queue
-------------------- ---------- --------------- ---------- ---------------
STREAMS_PROPAGATION  STRMADMIN  STREAMS_CAPTURE STRMADMIN  STREAMS_APPLY_Q
                                _Q@INST1.NET               @INST2.NET
```

Determining the Rule Sets for Each Propagation

The query in this section displays the following information for each **propagation**:

- The propagation name
- The owner of the **positive rule set** for the propagation
- The name of the positive rule set used by the propagation
- The owner of the **negative rule set** used by the propagation
- The name of the negative rule set used by the propagation

To display this general information about each propagation in a database, run the following query:

```
COLUMN PROPAGATION_NAME HEADING 'Propagation|Name' FORMAT A20
COLUMN RULE_SET_OWNER HEADING 'Positive|Rule Set|Owner' FORMAT A10
COLUMN RULE_SET_NAME HEADING 'Positive Rule|Set Name' FORMAT A15
COLUMN NEGATIVE_RULE_SET_OWNER HEADING 'Negative|Rule Set|Owner' FORMAT A10
COLUMN NEGATIVE_RULE_SET_NAME HEADING 'Negative Rule|Set Name' FORMAT A15

SELECT PROPAGATION_NAME,
       RULE_SET_OWNER,
       RULE_SET_NAME,
       NEGATIVE_RULE_SET_OWNER,
       NEGATIVE_RULE_SET_NAME
  FROM DBA_PROPAGATION;
```

Your output looks similar to the following:

```
                    Positive                    Negative
Propagation         Rule Set   Positive Rule    Rule Set   Negative Rule
Name                Owner      Set Name         Owner      Set Name
------------------- ---------- ---------------- ---------- ---------------
STRM01_PROPAGATION  STRMADMIN  RULESET$_22      STRMADMIN  RULESET$_31
```

Displaying the Schedule for a Propagation Job

The query in this section displays the following information about the **propagation schedule** for a **propagation job** used by a **propagation** named `dbs1_to_dbs2`:

- The date and time when the propagation schedule started (or will start).
- The duration of the propagation job, which is the amount of time the job propagates **message**s before restarting.
- The next time the propagation will start.
- The latency of the propagation job, which is the maximum wait time to propagate a new message during the duration, when all other messages in the **queue** to the relevant destination have been propagated.
- Whether or not the propagation job is enabled.
- The name of the process that most recently executed the schedule.
- The number of consecutive times schedule execution has failed, if any. After 16 consecutive failures, a propagation job becomes disabled automatically.

Run this query at the database that contains the **source queue**:

```
COLUMN START_DATE HEADING 'Start Date'
COLUMN PROPAGATION_WINDOW HEADING 'Duration|in Seconds' FORMAT 99999
COLUMN NEXT_TIME HEADING 'Next|Time' FORMAT A8
COLUMN LATENCY HEADING 'Latency|in Seconds' FORMAT 99999
COLUMN SCHEDULE_DISABLED HEADING 'Status' FORMAT A8
COLUMN PROCESS_NAME HEADING 'Process' FORMAT A8
COLUMN FAILURES HEADING 'Number of|Failures' FORMAT 99

SELECT DISTINCT TO_CHAR(s.START_DATE, 'HH24:MI:SS MM/DD/YY') START_DATE,
       s.PROPAGATION_WINDOW,
       s.NEXT_TIME,
       s.LATENCY,
       DECODE(s.SCHEDULE_DISABLED,
                'Y', 'Disabled',
                'N', 'Enabled') SCHEDULE_DISABLED,
       s.PROCESS_NAME,
       s.FAILURES
  FROM DBA_QUEUE_SCHEDULES s, DBA_PROPAGATION p
  WHERE p.PROPAGATION_NAME = 'DBS1_TO_DBS2'
  AND p.DESTINATION_DBLINK = s.DESTINATION
  AND s.SCHEMA = p.SOURCE_QUEUE_OWNER
  AND s.QNAME = p.SOURCE_QUEUE_NAME;
```

Your output looks similar to the following:

```
                   Duration Next        Latency                   Number of
Start Date        in Seconds Time     in Seconds Status   Process   Failures
----------------- ---------- -------- ---------- -------- -------- ---------
15:23:40 03/02/02                             5 Enabled  J002             0
```

This propagation job uses the default schedule for a Streams propagation job. That is, the duration and next time are both NULL, and the latency is five seconds. When the duration is NULL, the job propagates changes without restarting automatically. When the next time is NULL, the propagation job is running currently.

See Also:

- "Propagation Scheduling and Streams Propagations" on page 3-22 for more information about the default propagation schedule for a Streams propagation job
- "Is the Propagation Enabled?" on page 18-7 if the propagation job is disabled
- *Oracle Streams Advanced Queuing User's Guide and Reference* and *Oracle Database Reference* for more information about the DBA_QUEUE_SCHEDULES data dictionary view

Determining the Total Number of Messages and Bytes Propagated

All **propagation job**s from a **source queue** that share the same database link have a single **propagation schedule**. The query in this section displays the following information for each **propagation**:

- The name of the propagation
- The total time spent by the system executing the propagation schedule
- The total number of **message**s propagated by the propagation schedule
- The total number of bytes propagated by the propagation schedule

Run the following query to display this information for each propagation with a source queue at the local database:

```
COLUMN PROPAGATION_NAME HEADING 'Propagation|Name' FORMAT A20
COLUMN TOTAL_TIME HEADING 'Total Time|Executing|in Seconds' FORMAT 999999
COLUMN TOTAL_NUMBER HEADING 'Total Messages|Propagated' FORMAT 999999999
COLUMN TOTAL_BYTES HEADING 'Total Bytes|Propagated' FORMAT 9999999999999

SELECT p.PROPAGATION_NAME, s.TOTAL_TIME, s.TOTAL_NUMBER, s.TOTAL_BYTES
  FROM DBA_QUEUE_SCHEDULES s, DBA_PROPAGATION p
  WHERE p.DESTINATION_DBLINK = s.DESTINATION
    AND s.SCHEMA = p.SOURCE_QUEUE_OWNER
    AND s.QNAME = p.SOURCE_QUEUE_NAME;
```

Your output looks similar to the following:

```
                    Total Time
Propagation          Executing Total Messages    Total Bytes
Name                in Seconds   Propagated       Propagated
------------------- ---------- -------------- --------------
MULT3_TO_MULT1              351            872         875252
MULT3_TO_MULT2              596            872         875252
```

See Also: *Oracle Streams Advanced Queuing User's Guide and Reference* and *Oracle Database Reference* for more information about the DBA_QUEUE_SCHEDULES data dictionary view

22

Monitoring Streams Apply Processes

This chapter provides sample queries that you can use to monitor your Streams **apply process**es.

This chapter contains these topics:

Note: The Streams tool in the Oracle Enterprise Manager Console is also an excellent way to monitor a Streams environment. See the online help for the Streams tool for more information.

See Also:

- Chapter 4, "Streams Apply Process"
- Chapter 13, "Managing an Apply Process"
- *Oracle Database Reference* for information about the data dictionary views described in this chapter
- *Oracle Streams Replication Administrator's Guide* for information about monitoring a Streams **replication** environment

Determining the Queue, Rule Sets, and Status for Each Apply Process

You can determine the following information for each **apply process** in a database by running the query in this section:

- The apply process name
- The name of the **queue** used by the apply process
- The name of the **positive rule set** used by the apply process
- The name of the **negative rule set** used by the apply process
- The status of the apply process, either ENABLED, DISABLED, or ABORTED

To display this general information about each apply process in a database, run the following query:

```
COLUMN APPLY_NAME HEADING 'Apply|Process|Name' FORMAT A15
COLUMN QUEUE_NAME HEADING 'Apply|Process|Queue' FORMAT A15
COLUMN RULE_SET_NAME HEADING 'Positive|Rule Set' FORMAT A15
COLUMN NEGATIVE_RULE_SET_NAME HEADING 'Negative|Rule Set' FORMAT A15
COLUMN STATUS HEADING 'Apply|Process|Status' FORMAT A15

SELECT APPLY_NAME,
       QUEUE_NAME,
       RULE_SET_NAME,
       NEGATIVE_RULE_SET_NAME,
       STATUS
  FROM DBA_APPLY;
```

Your output looks similar to the following:

```
Apply           Apply                                          Apply
Process         Process         Positive        Negative       Process
Name            Queue           Rule Set        Rule Set       Status
--------------- --------------- --------------- --------------- ---------------
STRM01_APPLY    STRM01_QUEUE    RULESET$_36                    ENABLED
APPLY_EMP       STREAMS_QUEUE   RULESET$_16                    DISABLED
APPLY           STREAMS_QUEUE   RULESET$_21     RULESET$_23    ENABLED
```

If the status of an apply process is ABORTED, then you can query the ERROR_NUMBER and ERROR_MESSAGE columns in the DBA_APPLY data dictionary view to determine the error. These columns are populated when an apply process aborts or when an apply process is disabled after reaching a limit. These columns are cleared when an apply process is restarted.

Note: The ERROR_NUMBER and ERROR_MESSAGE columns in the DBA_APPLY data dictionary view are not related to the information in the DBA_APPLY_ERROR data dictionary view.

See Also: "Checking for Apply Errors" on page 22-15 to check for apply errors if the apply process status is ABORTED

Displaying General Information About Each Apply Process

You can display the following general information about each **apply process** in a database by running the query in this section:

- The apply process name.
- The type of **message**s applied by the apply process. An apply process can apply either messages that were captured by a **capture process** or messages that were enqueued by a user or application.
- The **apply user**.

To display this general information about each apply process in a database, run the following query:

```
COLUMN APPLY_NAME HEADING 'Apply Process Name' FORMAT A20
COLUMN APPLY_CAPTURED HEADING 'Type of Messages Applied' FORMAT A25
COLUMN APPLY_USER HEADING 'Apply User' FORMAT A30

SELECT APPLY_NAME,
       DECODE(APPLY_CAPTURED,
              'YES', 'Captured',
              'NO',  'User-Enqueued') APPLY_CAPTURED,
       APPLY_USER
  FROM DBA_APPLY;
```

Your output looks similar to the following:

```
Apply Process Name   Type of Messages Applied  Apply User
-------------------- ------------------------- ------------------------------
STRM01_APPLY         Captured                  STRMADMIN
APPLY_OE             User-Enqueued             STRMADMIN
APPLY                Captured                  HR
```

Listing the Parameter Settings for Each Apply Process

The following query displays the current setting for each **apply process** parameter for each apply process in a database:

```
COLUMN APPLY_NAME HEADING 'Apply Process|Name' FORMAT A15
COLUMN PARAMETER HEADING 'Parameter' FORMAT A25
COLUMN VALUE HEADING 'Value' FORMAT A20
COLUMN SET_BY_USER HEADING 'Set by User?' FORMAT A15

SELECT APPLY_NAME,
       PARAMETER,
       VALUE,
       SET_BY_USER
  FROM DBA_APPLY_PARAMETERS;
```

Your output looks similar to the following:

```
Apply Process
Name            Parameter                 Value                Set by User?
--------------- ------------------------- -------------------- ---------------
APPLY_HR        ALLOW_DUPLICATE_ROWS      N                    NO
APPLY_HR        COMMIT_SERIALIZATION      FULL                 NO
APPLY_HR        DISABLE_ON_ERROR          Y                    NO
APPLY_HR        DISABLE_ON_LIMIT          N                    NO
APPLY_HR        MAXIMUM_SCN               INFINITE             NO
APPLY_HR        PARALLELISM               1                    NO
APPLY_HR        STARTUP_SECONDS           0                    NO
APPLY_HR        TIME_LIMIT                INFINITE             NO
APPLY_HR        TRACE_LEVEL               0                    NO
APPLY_HR        TRANSACTION_LIMIT         INFINITE             NO
APPLY_HR        TXN_LCR_SPILL_THRESHOLD   5000                 YES
APPLY_HR        WRITE_ALERT_LOG           Y                    NO
```

Note: If the `Set by User?` column is `NO` for a parameter, then the parameter is set to its default value. If the `Set by User?` column is `YES` for a parameter, then the parameter might or might not be set to its default value.

See Also:

- "Apply Process Parameters" on page 4-14
- "Setting an Apply Process Parameter" on page 13-11

Displaying Information About Apply Handlers

This section contains instructions for displaying information about **apply process message handler**s and **error handler**s.

See Also:

- "Message Processing with an Apply Process" on page 4-2
- *Oracle Streams Replication Administrator's Guide* for information about monitoring **DML handler**s and **DDL handler**s

Displaying All of the Error Handlers for Local Apply Processes

When you specify a local **error handler** using the `SET_DML_HANDLER` procedure in the `DBMS_APPLY_ADM` package at a **destination database**, you can specify either that the handler runs for a specific apply process or that the handler is a general handler that runs for all apply processes in the database that apply changes locally when an error is raised by an apply process. A specific error handler takes precedence over a generic error handler. An error handler is run for a specified operation on a specific table.

To display the error handler for each apply process that applies changes locally in a database, run the following query:

```
COLUMN OBJECT_OWNER HEADING 'Table|Owner' FORMAT A5
COLUMN OBJECT_NAME HEADING 'Table Name' FORMAT A10
COLUMN OPERATION_NAME HEADING 'Operation' FORMAT A10
COLUMN USER_PROCEDURE HEADING 'Handler Procedure' FORMAT A30
COLUMN APPLY_NAME HEADING 'Apply Process|Name' FORMAT A15

SELECT OBJECT_OWNER,
       OBJECT_NAME,
       OPERATION_NAME,
       USER_PROCEDURE,
       APPLY_NAME
  FROM DBA_APPLY_DML_HANDLERS
  WHERE ERROR_HANDLER = 'Y'
  ORDER BY OBJECT_OWNER, OBJECT_NAME;
```

Your output looks similar to the following:

```
Table                                                          Apply Process
Owner Table Name Operation  Handler Procedure                  Name
----- ---------- ---------- ------------------------------ --------------
HR    REGIONS    INSERT     "STRMADMIN"."ERRORS_PKG"."REGI
                            ONS_PK_ERROR"
```

`Apply Process Name` is `NULL` for the `strmadmin.errors_pkg.regions_pk_error` error handler. Therefore, this handler is a general handler that runs for all of the local apply processes.

See Also: "Managing an Error Handler" on page 13-18

Displaying the Message Handler for Each Apply Process

To display each **message handler** in a database, run the following query:

```
COLUMN APPLY_NAME HEADING 'Apply Process Name' FORMAT A20
COLUMN MESSAGE_HANDLER HEADING 'Message Handler' FORMAT A20

SELECT APPLY_NAME, MESSAGE_HANDLER FROM DBA_APPLY
  WHERE MESSAGE_HANDLER IS NOT NULL;
```

Your output looks similar to the following:

```
Apply Process Name   Message Handler
-------------------- --------------------
STRM03_APPLY         "OE"."MES_HANDLER"
```

See Also: "Managing the Message Handler for an Apply Process" on page 13-12

Displaying the Precommit Handler for Each Apply Process

To display each **precommit handler** in a database, run the following query:

```
COLUMN APPLY_NAME HEADING 'Apply Process Name' FORMAT A20
COLUMN PRECOMMIT_HANDLER HEADING 'Precommit Handler' FORMAT A30
COLUMN APPLY_CAPTURED HEADING 'Type of|Messages|Applied' FORMAT A15
```

```
SELECT APPLY_NAME,
       PRECOMMIT_HANDLER,
       DECODE(APPLY_CAPTURED,
              'YES', 'Captured',
              'NO',  'User-Enqueued') APPLY_CAPTURED
  FROM DBA_APPLY
  WHERE PRECOMMIT_HANDLER IS NOT NULL;
```

Your output looks similar to the following:

```
                                                    Type of
                                                    Messages
Apply Process Name   Precommit Handler              Applied
-------------------- ------------------------------ ---------------
STRM01_APPLY         "STRMADMIN"."HISTORY_COMMIT"   Captured
```

See Also: "Managing the Precommit Handler for an Apply Process" on page 13-13

Displaying Information About the Reader Server for Each Apply Process

The **reader server** for an **apply process** dequeues **message**s from the **queue**. The reader server is a parallel execution server that computes dependencies between LCRs and assembles messages into transactions. The reader server then returns the assembled transactions to the coordinator, which assigns them to idle **apply servers**.

The query in this section displays the following information about the reader server for each apply process:

- The name of the apply process
- The type of messages dequeued by the reader server, either **captured message**s or **user-enqueued message**s
- The name of the parallel execution server used by the reader server
- The current state of the reader server, either `INITIALIZING`, `IDLE`, `DEQUEUE MESSAGES`, `SCHEDULE MESSAGES`, `SPILLING`, or `PAUSED`
- The total number of messages dequeued by the reader server since the last time the apply process was started

The information displayed by this query is valid only for an enabled apply process.

Run the following query to display this information for each apply process:

```
COLUMN APPLY_NAME HEADING 'Apply Process|Name' FORMAT A15
COLUMN APPLY_CAPTURED HEADING 'Apply Type' FORMAT A22
COLUMN PROCESS_NAME HEADING 'Process|Name' FORMAT A7
COLUMN STATE HEADING 'State' FORMAT A17
COLUMN TOTAL_MESSAGES_DEQUEUED HEADING 'Total Messages|Dequeued' FORMAT 99999999
```

```
SELECT r.APPLY_NAME,
       DECODE(ap.APPLY_CAPTURED,
                'YES','Captured LCRS',
                'NO','User-enqueued messages','UNKNOWN') APPLY_CAPTURED,
       SUBSTR(s.PROGRAM,INSTR(s.PROGRAM,'(')+1,4) PROCESS_NAME,
       r.STATE,
       r.TOTAL_MESSAGES_DEQUEUED
       FROM V$STREAMS_APPLY_READER r, V$SESSION s, DBA_APPLY ap
       WHERE r.SID = s.SID AND
             r.SERIAL# = s.SERIAL# AND
             r.APPLY_NAME = ap.APPLY_NAME;
```

Your output looks similar to the following:

```
Apply Process                          Process                  Total Messages
Name            Apply Type             Name    State                  Dequeued
--------------- ---------------------- ------- ----------------- --------------
APPLY$_STM2_14  Captured LCRS          P000    DEQUEUE MESSAGES            5650
```

See Also: "Reader Server States" on page 4-11

Monitoring Transactions and Messages Spilled by Each Apply Process

If the `txn_lcr_spill_threshold` **apply process** parameter is set to a value other than `infinite`, then an apply process can spill **message**s from memory to hard disk when the number of messages in a transaction exceeds the specified number.

The first query in this section displays the following information about each transaction currently being applied for which the apply process has spilled messages:

- The name of the apply process
- The transaction ID of the transaction with spilled messages
- The system change number (SCN) of the first message in the transaction
- The number of messages currently spilled in the transaction

To display this information for each apply process in a database, run the following query:

```
COLUMN APPLY_NAME HEADING 'Apply Name' FORMAT A20
COLUMN 'Transaction ID' HEADING 'Transaction ID' FORMAT A15
COLUMN FIRST_SCN HEADING 'First SCN'   FORMAT 99999999
COLUMN MESSAGE_COUNT HEADING 'Message Count' FORMAT 99999999

SELECT APPLY_NAME,
       XIDUSN ||'.'||
       XIDSLT ||'.'||
       XIDSQN "Transaction ID",
       FIRST_SCN,
       MESSAGE_COUNT
  FROM DBA_APPLY_SPILL_TXN;
```

Your output looks similar to the following:

```
Apply Name           Transaction ID  First SCN Message Count
-------------------- --------------- --------- -------------
APPLY_HR             1.42.2277         2246944           100
```

The next query in this section displays the following information about the messages spilled by the apply processes in the local database:

- The name of the apply process
- The total number of messages spilled by the apply process since it last started
- The amount of time the apply process spent spilling messages, in seconds

To display this information for each apply process in a database, run the following query:

```
COLUMN APPLY_NAME HEADING 'Apply Name' FORMAT A15
COLUMN TOTAL_MESSAGES_SPILLED HEADING 'Total|Spilled Messages' FORMAT 99999999
COLUMN ELAPSED_SPILL_TIME HEADING 'Elapsed Time|Spilling Messages' FORMAT
99999999.99

SELECT APPLY_NAME,
       TOTAL_MESSAGES_SPILLED,
       (ELAPSED_SPILL_TIME/100) ELAPSED_SPILL_TIME
  FROM V$STREAMS_APPLY_READER;
```

Your output looks similar to the following:

```
                          Total      Elapsed Time
Apply Name      Spilled Messages Spilling Messages
--------------- ---------------- -----------------
APPLY_HR                    100              2.67
```

Note: The elapsed time spilling messages is displayed in seconds. The `V$STREAMS_APPLY_READER` view displays elapsed time in centiseconds by default. A centisecond is one-hundredth of a second. The query in this section divides each elapsed time by one hundred to display the elapsed time in seconds.

Determining Capture to Dequeue Latency for a Message

The query in this section displays the following information about the last **message** dequeued by each **apply process**:

- The name of the apply process.
- The latency. For **captured message**s, the latency is the amount of time between when the message was created at a **source database** and when the message was dequeued by the apply process. For **user-enqueued message**s, the latency is the amount of time between when the message enqueued at the local database and when the message was dequeued by the apply process.
- The message creation time. For captured messages, the message creation time is the time when the data manipulation language (DML) or data definition language (DDL) change generated the redo data at the source database for the message. For user-enqueued messages, the message creation time is the last time the message was enqueued. A user-enqueued message can be enqueued one or more additional times by **propagation**s before it reaches an apply process.
- The time when the message was dequeued by the apply process.
- The message number of the message that was last dequeued by the apply process.

The information displayed by this query is valid only for an enabled apply process.

Run the following query to display this information for each apply process:

```
COLUMN APPLY_NAME HEADING 'Apply Process|Name' FORMAT A17
COLUMN LATENCY HEADING 'Latency|in|Seconds' FORMAT 9999
COLUMN CREATION HEADING 'Message Creation' FORMAT A17
COLUMN LAST_DEQUEUE HEADING 'Last Dequeue Time' FORMAT A20
COLUMN DEQUEUED_MESSAGE_NUMBER HEADING 'Dequeued|Message Number' FORMAT 999999

SELECT APPLY_NAME,
     (DEQUEUE_TIME-DEQUEUED_MESSAGE_CREATE_TIME)*86400 LATENCY,
     TO_CHAR(DEQUEUED_MESSAGE_CREATE_TIME,'HH24:MI:SS MM/DD/YY') CREATION,
     TO_CHAR(DEQUEUE_TIME,'HH24:MI:SS MM/DD/YY') LAST_DEQUEUE,
     DEQUEUED_MESSAGE_NUMBER
  FROM V$STREAMS_APPLY_READER;
```

Your output looks similar to the following:

```
                 Latency
Apply Process         in                                          Dequeued
Name             Seconds Message Creation  Last Dequeue Time    Message Number
---------------- ------- ----------------- -------------------- --------------
APPLY$_STM1_14         1 15:22:15 06/13/05 15:22:16 06/13/05            502129
```

Displaying General Information About Each Coordinator Process

A **coordinator process** gets transactions from the **reader server** and passes these transactions to **apply servers**. The **coordinator process** name is `ap`*nn*, where *nn* is a coordinator process number.

The query in this section displays the following information about the coordinator process for each **apply process**:

- The apply process name
- The number of the coordinator in the process name (`ap`*nn*)
- The session identifier of the coordinator's session
- The serial number of the coordinator's session
- The current state of the coordinator, either `INITIALIZING`, `APPLYING`, `SHUTTING DOWN CLEANLY`, or `ABORTING`

The information displayed by this query is valid only for an enabled apply process.

Run the following query to display this information for each apply process:

```
COLUMN APPLY_NAME HEADING 'Apply Process|Name' FORMAT A17
COLUMN PROCESS_NAME HEADING 'Coordinator|Process|Name' FORMAT A11
COLUMN SID HEADING 'Session|ID' FORMAT 9999
COLUMN SERIAL# HEADING 'Session|Serial|Number' FORMAT 9999
COLUMN STATE HEADING 'State' FORMAT A21

SELECT c.APPLY_NAME,
       SUBSTR(s.PROGRAM,INSTR(s.PROGRAM,'(')+1,4) PROCESS_NAME,
       c.SID,
       c.SERIAL#,
       c.STATE
       FROM V$STREAMS_APPLY_COORDINATOR c, V$SESSION s
       WHERE c.SID = s.SID AND
             c.SERIAL# = s.SERIAL#;
```

Your output looks similar to the following:

```
                 Coordinator         Session
Apply Process    Process     Session  Serial
Name             Name             ID  Number State
---------------- ----------- ------- ------- --------------------
APPLY_FROM_MULT1 A001             16       1 APPLYING
APPLY_FROM_MULT2 A002             18       1 APPLYING
```

See Also: "Coordinator Process States" on page 4-12

Displaying Information About Transactions Received and Applied

The query in this section displays the following information about the transactions received, applied, and being applied by each **apply process**:

- The apply process name
- The total number of transactions received by the **coordinator process** since the apply process was last started
- The total number of transactions successfully applied by the apply process since the apply process was last started
- The number of transactions applied by the apply process that resulted in an apply error since the apply process was last started
- The total number of transactions currently being applied by the apply process
- The total number of transactions received by the coordinator process but ignored because the apply process had already applied the transactions since the apply process was last started

The information displayed by this query is valid only for an enabled apply process.

For example, to display this information for an apply process named `apply`, run the following query:

```
COLUMN APPLY_NAME HEADING 'Apply Process Name' FORMAT A25
COLUMN TOTAL_RECEIVED HEADING 'Total|Trans|Received' FORMAT 99999999
COLUMN TOTAL_APPLIED HEADING 'Total|Trans|Applied' FORMAT 99999999
COLUMN TOTAL_ERRORS HEADING 'Total|Apply|Errors' FORMAT 9999
COLUMN BEING_APPLIED HEADING 'Total|Trans Being|Applied' FORMAT 99999999
COLUMN TOTAL_IGNORED HEADING 'Total|Trans|Ignored' FORMAT 99999999

SELECT APPLY_NAME,
       TOTAL_RECEIVED,
       TOTAL_APPLIED,
       TOTAL_ERRORS,
       (TOTAL_ASSIGNED - (TOTAL_ROLLBACKS + TOTAL_APPLIED)) BEING_APPLIED,
       TOTAL_IGNORED
       FROM V$STREAMS_APPLY_COORDINATOR;
```

Your output looks similar to the following:

```
                           Total     Total  Total       Total     Total
                           Trans     Trans  Apply Trans Being     Trans
Apply Process Name      Received   Applied Errors     Applied   Ignored
------------------------ --------- --------- ------ ----------- ---------
APPLY_FROM_MULT1               81        73      2           6         0
APPLY_FROM_MULT2              114        96      0          14         4
```

Determining the Capture to Apply Latency for a Message for Each Apply Process

This section contains two different queries that show the capture to apply latency for a particular **message**. That is, for **captured message**s, these queries show the amount of time between when the message was created at a **source database** and when the message was applied by the **apply process**. One query uses the V$STREAMS_APPLY_COORDINATOR dynamic performance view. The other uses the DBA_APPLY_PROGRESS static data dictionary view.

> **Note:** These queries assume that the apply process applies captured messages, not user-enqueued messages.

The two **queue**s differ in the following ways:

- The apply process must be enabled when you run the query on the V$STREAMS_APPLY_COORDINATOR view, while the apply process can be enabled or disabled when you run the query on the DBA_APPLY_PROGRESS view.
- The query on the V$STREAMS_APPLY_COORDINATOR view can show the latency for a more recent transaction than the query on the DBA_APPLY_PROGRESS view.

Both queries display the following information about a message applied by each apply process:

- The apply process name.
- The capture to apply latency for the message.
- The message creation time. For captured messages, the message creation time is the time when the data manipulation language (DML) or data definition language (DDL) change generated the redo data at the source database for the message.
- The time when the message was applied by the apply process.
- The message number of the message.

Example V$STREAMS_APPLY_COORDINATOR Query for Latency

Run the following query to display the capture to apply latency using the V$STREAMS_APPLY_COORDINATOR view for a **message** for each apply process:

```
COLUMN APPLY_NAME HEADING 'Apply Process|Name' FORMAT A17
COLUMN 'Latency in Seconds' FORMAT 999999
COLUMN 'Message Creation' FORMAT A17
COLUMN 'Apply Time' FORMAT A17
COLUMN HWM_MESSAGE_NUMBER HEADING 'Applied|Message|Number' FORMAT 999999

SELECT APPLY_NAME,
     (HWM_TIME-HWM_MESSAGE_CREATE_TIME)*86400 "Latency in Seconds",
     TO_CHAR(HWM_MESSAGE_CREATE_TIME,'HH24:MI:SS MM/DD/YY')
        "Message Creation",
     TO_CHAR(HWM_TIME,'HH24:MI:SS MM/DD/YY') "Apply Time",
     HWM_MESSAGE_NUMBER
  FROM V$STREAMS_APPLY_COORDINATOR;
```

Your output looks similar to the following:

```
                                                                                Applied
Apply Process                                                                   Message
Name              Latency in Seconds Message Creation  Apply Time          Number
----------------- ------------------ ----------------- ----------------- -------
APPLY$_STM1_14                     4 14:05:13 06/13/05 14:05:17 06/13/05  498215
```

Example DBA_APPLY_PROGRESS Query for Latency

Run the following query to display the capture to apply latency using the DBA_APPLY_PROGRESS view for a **message** for each apply process:

```
COLUMN APPLY_NAME HEADING 'Apply Process|Name' FORMAT A17
COLUMN 'Latency in Seconds' FORMAT 999999
COLUMN 'Message Creation' FORMAT A17
COLUMN 'Apply Time' FORMAT A17
COLUMN APPLIED_MESSAGE_NUMBER HEADING 'Applied|Message|Number' FORMAT 999999

SELECT APPLY_NAME,
     (APPLY_TIME-APPLIED_MESSAGE_CREATE_TIME)*86400 "Latency in Seconds",
     TO_CHAR(APPLIED_MESSAGE_CREATE_TIME,'HH24:MI:SS MM/DD/YY')
        "Message Creation",
     TO_CHAR(APPLY_TIME,'HH24:MI:SS MM/DD/YY') "Apply Time",
     APPLIED_MESSAGE_NUMBER
  FROM DBA_APPLY_PROGRESS;
```

Your output looks similar to the following:

```
                                                                                Applied
Apply Process                                                                   Message
Name              Latency in Seconds Message Creation  Apply Time          Number
----------------- ------------------ ----------------- ----------------- -------
APPLY$_STM1_14                    33 14:05:13 06/13/05 14:05:46 06/13/05  498215
```

Displaying Information About the Apply Servers for Each Apply Process

An **apply process** can use one or more **apply servers** that apply LCRs to database objects as DML statements or DDL statements or pass the LCRs to their appropriate handlers. For non-LCR **message**s, the apply servers pass the messages to the **message handler**. Each apply server is a parallel execution server.

The query in this section displays the following information about the apply servers for each apply process:

- The name of the apply process.
- The process names of the parallel execution servers, in order.
- The current state of each apply server:
  - IDLE
  - RECORD LOW-WATERMARK
  - ADD PARTITION
  - DROP PARTITION
  - EXECUTE TRANSACTION
  - WAIT COMMIT
  - WAIT DEPENDENCY

  - WAIT FOR NEXT CHUNK
  - TRANSACTION CLEANUP

- The total number of transactions assigned to each apply server since the last time the apply process was started. A transaction can contain more than one message.
- The total number of messages applied by each apply server since the last time the apply process was started.

The information displayed by this query is valid only for an enabled apply process.

Run the following query to display information about the apply servers for each apply process:

```
COLUMN APPLY_NAME HEADING 'Apply Process Name' FORMAT A22
COLUMN PROCESS_NAME HEADING 'Process Name' FORMAT A12
COLUMN STATE HEADING 'State' FORMAT A17
COLUMN TOTAL_ASSIGNED HEADING 'Total|Transactions|Assigned' FORMAT 99999999
COLUMN TOTAL_MESSAGES_APPLIED HEADING 'Total|Messages|Applied' FORMAT 99999999

SELECT r.APPLY_NAME,
       SUBSTR(s.PROGRAM,INSTR(S.PROGRAM,'(')+1,4) PROCESS_NAME,
       r.STATE,
       r.TOTAL_ASSIGNED,
       r.TOTAL_MESSAGES_APPLIED
  FROM V$STREAMS_APPLY_SERVER R, V$SESSION S
  WHERE r.SID = s.SID AND
        r.SERIAL# = s.SERIAL#
  ORDER BY r.APPLY_NAME, r.SERVER_ID;
```

Your output looks similar to the following:

```
                                                              Total      Total
                                                          Transactions   Messages
Apply Process Name     Process Name State                     Assigned    Applied
---------------------- ------------ ----------------- ------------ ----------
APPLY                  P001         IDLE                         94       2141
APPLY                  P002         IDLE                         12        276
APPLY                  P003         IDLE                          0          0
```

See Also: "Apply Server States" on page 4-12

Displaying Effective Apply Parallelism for an Apply Process

In some environments, an **apply process** might not use all of the **apply servers** available to it. For example, apply process parallelism can be set to five, but only three apply servers are ever used by the apply process. In this case, the effective apply parallelism is three.

The following query displays the effective apply parallelism for an apply process named `apply`:

```
SELECT COUNT(SERVER_ID) "Effective Parallelism"
  FROM V$STREAMS_APPLY_SERVER
  WHERE APPLY_NAME = 'APPLY' AND
        TOTAL_MESSAGES_APPLIED > 0;
```

Your output looks similar to the following:

```
Effective Parallelism
---------------------
                    2
```

This query returned two for the effective parallelism. If parallelism is set to three for the apply process named `apply`, then one apply server has not been used since the last time the apply process was started.

You can display the total number of **message**s applied by each apply server by running the following query:

```
COLUMN SERVER_ID HEADING 'Apply Server ID' FORMAT 99
COLUMN TOTAL_MESSAGES_APPLIED HEADING 'Total Messages Applied' FORMAT 999999

SELECT SERVER_ID, TOTAL_MESSAGES_APPLIED
  FROM V$STREAMS_APPLY_SERVER
  WHERE APPLY_NAME = 'APPLY'
  ORDER BY SERVER_ID;
```

Your output looks similar to the following:

```
Apply Server ID Total Messages Applied
--------------- ----------------------
              1                   2141
              2                    276
              3                      0
```

In this case, apply server 3 has not been used by the apply process since it was last started. If the `parallelism` setting for an apply process is higher than the effective parallelism for the apply process, then consider lowering the `parallelism` setting.

Viewing Rules that Specify a Destination Queue on Apply

You can specify a **destination queue** for a **rule** using the `SET_ENQUEUE_DESTINATION` procedure in the `DBMS_APPLY_ADM` package. If an **apply process** has such a rule in its **positive rule set**, and a **message** satisfies the rule, then the apply process enqueues the message into the destination queue.

To view destination queue settings for rules, run the following query:

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

SELECT RULE_OWNER, RULE_NAME, DESTINATION_QUEUE_NAME
  FROM DBA_APPLY_ENQUEUE;
```

Your output looks similar to the following:

```
Rule Owner      Rule Name       Destination Queue
--------------- --------------- ------------------------------
STRMADMIN       DEPARTMENTS17   "STRMADMIN"."STREAMS_QUEUE"
```

See Also:

- "Specifying Message Enqueues by Apply Processes" on page 13-15
- "Enqueue Destinations for Messages During Apply" on page 6-39

Viewing Rules that Specify No Execution on Apply

You can specify an execution directive for a **rule** using the `SET_EXECUTE` procedure in the `DBMS_APPLY_ADM` package. An execution directive controls whether a **message** that satisfies the specified rule is executed by an **apply process**. If an apply process has a rule in its **positive rule set** with `NO` for its execution directive, and a message satisfies the rule, then the apply process does not execute the message and does not send the message to any **apply handler**.

To view each rule with `NO` for its execution directive, run the following query:

```
COLUMN RULE_OWNER HEADING 'Rule Owner' FORMAT A20
COLUMN RULE_NAME HEADING 'Rule Name' FORMAT A20

SELECT RULE_OWNER, RULE_NAME
  FROM DBA_APPLY_EXECUTE
  WHERE EXECUTE_EVENT = 'NO';
```

Your output looks similar to the following:

```
Rule Owner           Rule Name
-------------------- --------------------
STRMADMIN            DEPARTMENTS18
```

See Also:

- "Specifying Execute Directives for Apply Processes" on page 13-16
- "Execution Directives for Messages During Apply" on page 6-39

Checking for Apply Errors

To check for apply errors, run the following query:

```
COLUMN APPLY_NAME HEADING 'Apply|Process|Name' FORMAT A10
COLUMN SOURCE_DATABASE HEADING 'Source|Database' FORMAT A10
COLUMN LOCAL_TRANSACTION_ID HEADING 'Local|Transaction|ID' FORMAT A11
COLUMN ERROR_NUMBER HEADING 'Error Number' FORMAT 99999999
COLUMN ERROR_MESSAGE HEADING 'Error Message' FORMAT A20
COLUMN MESSAGE_COUNT HEADING 'Messages in|Error|Transaction' FORMAT 99999999

SELECT APPLY_NAME,
       SOURCE_DATABASE,
       LOCAL_TRANSACTION_ID,
       ERROR_NUMBER,
       ERROR_MESSAGE,
       MESSAGE_COUNT
  FROM DBA_APPLY_ERROR;
```

If there are any apply errors, then your output looks similar to the following:

```
Apply                Local                                        Messages in
Process    Source    Transaction                                        Error
Name       Database  ID          Error Number Error Message       Transaction
---------- --------- ----------- ------------ -------------------- -----------
APPLY_FROM MULT3.NET  1.62.948            1403 ORA-01403: no data f           1
_MULT3                                         ound

APPLY_FROM MULT2.NET  1.54.948            1403 ORA-01403: no data f           1
_MULT2                                         ound
```

If there are apply errors, then you can either try to reexecute the transactions that encountered the errors, or you can delete the transactions. If you want to reexecute a transaction that encountered an error, then first correct the condition that caused the transaction to raise an error.

If you want to delete a transaction that encountered an error, then you might need to resynchronize data manually if you are sharing data between multiple databases. Remember to set an appropriate session **tag**, if necessary, when you resynchronize data manually.

See Also:

- "The Error Queue" on page 4-16
- "Managing Apply Errors" on page 13-23
- *Oracle Streams Replication Administrator's Guide* for information about the possible causes of apply errors
- *Oracle Streams Replication Administrator's Guide* for more information about setting tag values generated by the current session

Displaying Detailed Information About Apply Errors

This section contains SQL scripts that you can use to display detailed information about the error transactions in the error queue in a database. These scripts are designed to display information about LCRs, but you can extend them to display information about any non-LCR **message**s used in your environment as well.

To use these scripts, complete the following steps:

1. Grant Explicit SELECT Privilege on the DBA_APPLY_ERROR View
2. Create a Procedure that Prints the Value in an ANYDATA Object
3. Create a Procedure that Prints a Specified LCR
4. Create a Procedure that Prints All the LCRs in the Error Queue
5. Create a Procedure that Prints All the Error LCRs for a Transaction

Note: These scripts display only the first 253 characters for `VARCHAR2` values in LCRs.

Step 1 Grant Explicit SELECT Privilege on the DBA_APPLY_ERROR View

The user who creates and runs the print_errors and print_transaction procedures described in the following sections must be granted explicit SELECT privilege on the DBA_APPLY_ERROR data dictionary view. This privilege cannot be granted through a role. Running the GRANT_ADMIN_PRIVILEGE procedure in the DBMS_STREAMS_AUTH package on a user grants this privilege to the user.

To grant this privilege to a user directly, complete the following steps:

1. Connect as an administrative user who can grant privileges.
2. Grant SELECT privilege on the DBA_APPLY_ERROR data dictionary view to the appropriate user. For example, to grant this privilege to the strmadmin user, run the following statement:

   ```
   GRANT SELECT ON DBA_APPLY_ERROR TO strmadmin;
   ```

3. Grant EXECUTE privilege on the DBMS_APPLY_ADM package. For example, to grant this privilege to the strmadmin user, run the following statement:

   ```
   GRANT EXECUTE ON DBMS_APPLY_ADM TO strmadmin;
   ```

4. Connect to the database as the user to whom you granted the privilege in Step 2 and 3.

Step 2 Create a Procedure that Prints the Value in an ANYDATA Object

The following procedure prints the value in a specified ANYDATA object for some selected datatypes. You can add more datatypes to this procedure if you wish.

```
CREATE OR REPLACE PROCEDURE print_any(data IN ANYDATA) IS
  tn  VARCHAR2(61);
  str VARCHAR2(4000);
  chr VARCHAR2(1000);
  num NUMBER;
  dat DATE;
  rw  RAW(4000);
  res NUMBER;
BEGIN
  IF data IS NULL THEN
    DBMS_OUTPUT.PUT_LINE('NULL value');
    RETURN;
  END IF;
  tn := data.GETTYPENAME();
  IF tn = 'SYS.VARCHAR2' THEN
    res := data.GETVARCHAR2(str);
    DBMS_OUTPUT.PUT_LINE(SUBSTR(str,0,253));
  ELSIF tn = 'SYS.CHAR' then
    res := data.GETCHAR(chr);
    DBMS_OUTPUT.PUT_LINE(SUBSTR(chr,0,253));
  ELSIF tn = 'SYS.VARCHAR' THEN
    res := data.GETVARCHAR(chr);
    DBMS_OUTPUT.PUT_LINE(chr);
  ELSIF tn = 'SYS.NUMBER' THEN
    res := data.GETNUMBER(num);
    DBMS_OUTPUT.PUT_LINE(num);
  ELSIF tn = 'SYS.DATE' THEN
    res := data.GETDATE(dat);
    DBMS_OUTPUT.PUT_LINE(dat);
```

```
  ELSIF tn = 'SYS.RAW' THEN
    -- res := data.GETRAW(rw);
    -- DBMS_OUTPUT.PUT_LINE(SUBSTR(DBMS_LOB.SUBSTR(rw),0,253));
    DBMS_OUTPUT.PUT_LINE('BLOB Value');
  ELSIF tn = 'SYS.BLOB' THEN
    DBMS_OUTPUT.PUT_LINE('BLOB Found');
  ELSE
    DBMS_OUTPUT.PUT_LINE('typename is ' || tn);
  END IF;
END print_any;
/
```

Step 3 Create a Procedure that Prints a Specified LCR

The following procedure prints a specified LCR. It calls the `print_any` procedure created in "Create a Procedure that Prints the Value in an ANYDATA Object" on page 22-17.

```
CREATE OR REPLACE PROCEDURE print_lcr(lcr IN ANYDATA) IS
  typenm    VARCHAR2(61);
  ddllcr    SYS.LCR$_DDL_RECORD;
  proclcr   SYS.LCR$_PROCEDURE_RECORD;
  rowlcr    SYS.LCR$_ROW_RECORD;
  res       NUMBER;
  newlist   SYS.LCR$_ROW_LIST;
  oldlist   SYS.LCR$_ROW_LIST;
  ddl_text  CLOB;
  ext_attr  ANYDATA;
BEGIN
  typenm := lcr.GETTYPENAME();
  DBMS_OUTPUT.PUT_LINE('type name: ' || typenm);
  IF (typenm = 'SYS.LCR$_DDL_RECORD') THEN
    res := lcr.GETOBJECT(ddllcr);
    DBMS_OUTPUT.PUT_LINE('source database: ' ||
                         ddllcr.GET_SOURCE_DATABASE_NAME);
    DBMS_OUTPUT.PUT_LINE('owner: ' || ddllcr.GET_OBJECT_OWNER);
    DBMS_OUTPUT.PUT_LINE('object: ' || ddllcr.GET_OBJECT_NAME);
    DBMS_OUTPUT.PUT_LINE('is tag null: ' || ddllcr.IS_NULL_TAG);
    DBMS_LOB.CREATETEMPORARY(ddl_text, true);
    ddllcr.GET_DDL_TEXT(ddl_text);
    DBMS_OUTPUT.PUT_LINE('ddl: ' || ddl_text);
    -- Print extra attributes in DDL LCR
    ext_attr := ddllcr.GET_EXTRA_ATTRIBUTE('serial#');
      IF (ext_attr IS NOT NULL) THEN
        DBMS_OUTPUT.PUT_LINE('serial#: ' || ext_attr.ACCESSNUMBER());
      END IF;
    ext_attr := ddllcr.GET_EXTRA_ATTRIBUTE('session#');
      IF (ext_attr IS NOT NULL) THEN
        DBMS_OUTPUT.PUT_LINE('session#: ' || ext_attr.ACCESSNUMBER());
      END IF;
    ext_attr := ddllcr.GET_EXTRA_ATTRIBUTE('thread#');
      IF (ext_attr IS NOT NULL) THEN
        DBMS_OUTPUT.PUT_LINE('thread#: ' || ext_attr.ACCESSNUMBER());
      END IF;
    ext_attr := ddllcr.GET_EXTRA_ATTRIBUTE('tx_name');
      IF (ext_attr IS NOT NULL) THEN
        DBMS_OUTPUT.PUT_LINE('transaction name: ' || ext_attr.ACCESSVARCHAR2());
      END IF;
```

```
    ext_attr := ddllcr.GET_EXTRA_ATTRIBUTE('username');
      IF (ext_attr IS NOT NULL) THEN
        DBMS_OUTPUT.PUT_LINE('username: ' || ext_attr.ACCESSVARCHAR2());
      END IF;
    DBMS_LOB.FREETEMPORARY(ddl_text);
  ELSIF (typenm = 'SYS.LCR$_ROW_RECORD') THEN
    res := lcr.GETOBJECT(rowlcr);
    DBMS_OUTPUT.PUT_LINE('source database: ' ||
                         rowlcr.GET_SOURCE_DATABASE_NAME);
    DBMS_OUTPUT.PUT_LINE('owner: ' || rowlcr.GET_OBJECT_OWNER);
    DBMS_OUTPUT.PUT_LINE('object: ' || rowlcr.GET_OBJECT_NAME);
    DBMS_OUTPUT.PUT_LINE('is tag null: ' || rowlcr.IS_NULL_TAG);
    DBMS_OUTPUT.PUT_LINE('command_type: ' || rowlcr.GET_COMMAND_TYPE);
    oldlist := rowlcr.GET_VALUES('old');
    FOR i IN 1..oldlist.COUNT LOOP
      IF oldlist(i) IS NOT NULL THEN
        DBMS_OUTPUT.PUT_LINE('old(' || i || '): ' || oldlist(i).column_name);
        print_any(oldlist(i).data);
      END IF;
    END LOOP;
    newlist := rowlcr.GET_VALUES('new', 'n');
    FOR i in 1..newlist.count LOOP
      IF newlist(i) IS NOT NULL THEN
        DBMS_OUTPUT.PUT_LINE('new(' || i || '): ' || newlist(i).column_name);
        print_any(newlist(i).data);
      END IF;
    END LOOP;
    -- Print extra attributes in row LCR
    ext_attr := rowlcr.GET_EXTRA_ATTRIBUTE('row_id');
      IF (ext_attr IS NOT NULL) THEN
        DBMS_OUTPUT.PUT_LINE('row_id: ' || ext_attr.ACCESSUROWID());
      END IF;
    ext_attr := rowlcr.GET_EXTRA_ATTRIBUTE('serial#');
      IF (ext_attr IS NOT NULL) THEN
        DBMS_OUTPUT.PUT_LINE('serial#: ' || ext_attr.ACCESSNUMBER());
      END IF;
    ext_attr := rowlcr.GET_EXTRA_ATTRIBUTE('session#');
      IF (ext_attr IS NOT NULL) THEN
        DBMS_OUTPUT.PUT_LINE('session#: ' || ext_attr.ACCESSNUMBER());
      END IF;
    ext_attr := rowlcr.GET_EXTRA_ATTRIBUTE('thread#');
      IF (ext_attr IS NOT NULL) THEN
        DBMS_OUTPUT.PUT_LINE('thread#: ' || ext_attr.ACCESSNUMBER());
      END IF;
    ext_attr := rowlcr.GET_EXTRA_ATTRIBUTE('tx_name');
      IF (ext_attr IS NOT NULL) THEN
        DBMS_OUTPUT.PUT_LINE('transaction name: ' || ext_attr.ACCESSVARCHAR2());
      END IF;
    ext_attr := rowlcr.GET_EXTRA_ATTRIBUTE('username');
      IF (ext_attr IS NOT NULL) THEN
        DBMS_OUTPUT.PUT_LINE('username: ' || ext_attr.ACCESSVARCHAR2());
      END IF;
  ELSE
    DBMS_OUTPUT.PUT_LINE('Non-LCR Message with type ' || typenm);
  END IF;
END print_lcr;
/
```

Step 4 Create a Procedure that Prints All the LCRs in the Error Queue

The following procedure prints all of the LCRs in all of the error queues. It calls the `print_lcr` procedure created in "Create a Procedure that Prints a Specified LCR" on page 22-18.

```
CREATE OR REPLACE PROCEDURE print_errors IS
  CURSOR c IS
    SELECT LOCAL_TRANSACTION_ID,
           SOURCE_DATABASE,
           MESSAGE_NUMBER,
           MESSAGE_COUNT,
           ERROR_NUMBER,
           ERROR_MESSAGE
      FROM DBA_APPLY_ERROR
      ORDER BY SOURCE_DATABASE, SOURCE_COMMIT_SCN;
  i      NUMBER;
  txnid  VARCHAR2(30);
  source VARCHAR2(128);
  msgno  NUMBER;
  msgcnt NUMBER;
  errnum NUMBER := 0;
  errno  NUMBER;
  errmsg VARCHAR2(255);
  lcr    ANYDATA;
  r      NUMBER;
BEGIN
  FOR r IN c LOOP
    errnum := errnum + 1;
    msgcnt := r.MESSAGE_COUNT;
    txnid  := r.LOCAL_TRANSACTION_ID;
    source := r.SOURCE_DATABASE;
    msgno  := r.MESSAGE_NUMBER;
    errno  := r.ERROR_NUMBER;
    errmsg := r.ERROR_MESSAGE;
DBMS_OUTPUT.PUT_LINE('************************************************');
    DBMS_OUTPUT.PUT_LINE('----- ERROR #' || errnum);
    DBMS_OUTPUT.PUT_LINE('----- Local Transaction ID: ' || txnid);
    DBMS_OUTPUT.PUT_LINE('----- Source Database: ' || source);
    DBMS_OUTPUT.PUT_LINE('----Error in Message: '|| msgno);
    DBMS_OUTPUT.PUT_LINE('----Error Number: '||errno);
    DBMS_OUTPUT.PUT_LINE('----Message Text: '||errmsg);
    FOR i IN 1..msgcnt LOOP
      DBMS_OUTPUT.PUT_LINE('--message: ' || i);
        lcr := DBMS_APPLY_ADM.GET_ERROR_MESSAGE(i, txnid);
        print_lcr(lcr);
    END LOOP;
  END LOOP;
END print_errors;
/
```

To run this procedure after you create it, enter the following:

```
SET SERVEROUTPUT ON SIZE 1000000

EXEC print_errors
```

Step 5 Create a Procedure that Prints All the Error LCRs for a Transaction

The following procedure prints all the LCRs in the error queue for a particular transaction. It calls the `print_lcr` procedure created in "Create a Procedure that Prints a Specified LCR" on page 22-18.

```
CREATE OR REPLACE PROCEDURE print_transaction(ltxnid IN VARCHAR2) IS
  i      NUMBER;
  txnid  VARCHAR2(30);
  source VARCHAR2(128);
  msgno  NUMBER;
  msgcnt NUMBER;
  errno  NUMBER;
  errmsg VARCHAR2(128);
  lcr    ANYDATA;
BEGIN
  SELECT LOCAL_TRANSACTION_ID,
         SOURCE_DATABASE,
         MESSAGE_NUMBER,
         MESSAGE_COUNT,
         ERROR_NUMBER,
         ERROR_MESSAGE
      INTO txnid, source, msgno, msgcnt, errno, errmsg
      FROM DBA_APPLY_ERROR
      WHERE LOCAL_TRANSACTION_ID =  ltxnid;
  DBMS_OUTPUT.PUT_LINE('----- Local Transaction ID: ' || txnid);
  DBMS_OUTPUT.PUT_LINE('----- Source Database: ' || source);
  DBMS_OUTPUT.PUT_LINE('----Error in Message: '|| msgno);
  DBMS_OUTPUT.PUT_LINE('----Error Number: '||errno);
  DBMS_OUTPUT.PUT_LINE('----Message Text: '||errmsg);
  FOR i IN 1..msgcnt LOOP
  DBMS_OUTPUT.PUT_LINE('--message: ' || i);
    lcr := DBMS_APPLY_ADM.GET_ERROR_MESSAGE(i, txnid); -- gets the LCR
    print_lcr(lcr);
  END LOOP;
END print_transaction;
/
```

To run this procedure after you create it, pass to it the local transaction identifier of a error transaction. For example, if the local transaction identifier is `1.17.2485`, then enter the following:

```
SET SERVEROUTPUT ON SIZE 1000000

EXEC print_transaction('1.17.2485')
```

23

Monitoring Rules

This chapter provides sample queries that you can use to monitor **rule**s, **rule set**s, and **evaluation context**s.

This chapter contains these topics:

Note: The Streams tool in the Oracle Enterprise Manager Console is also an excellent way to monitor a Streams environment. See the online help for the Streams tool for more information.

See Also:

- Chapter 5, "Rules"
- Chapter 6, "How Rules Are Used in Streams"
- Chapter 14, "Managing Rules"
- "Modifying a Name-Value Pair in a Rule Action Context" on page 14-7 for information about viewing a rule **action context**
- *Oracle Database Reference* for information about the data dictionary views described in this chapter
- *Oracle Streams Replication Administrator's Guide* for information about monitoring a Streams **replication** environment

Displaying All Rules Used by All Streams Clients

Streams **rule**s are created using the `DBMS_STREAMS_ADM` package or the Streams tool in the Oracle Enterprise Manager Console. Streams rules in the **rule set**s for a **Streams client** determine the behavior of the Streams client. Streams clients include **capture process**es, **propagation**s, **apply process**es, and **messaging client**s. The rule sets for a Streams client can also contain rules created using the `DBMS_RULE_ADM` package, and these rules also determine the behavior of the Streams client.

For example, if a rule in the **positive rule set** for a capture process evaluates to `TRUE` for DML changes to the `hr.employees` table, then the capture process captures DML changes to this table. However, if a rule in the **negative rule set** for a capture process evaluates to `TRUE` for DML changes to the `hr.employees` table, then the capture process discards DML changes to this table.

You query the following data dictionary views to display all rules in the rule sets for Streams clients, including Streams rules and rules created using the `DBMS_RULE_ADM` package:

- `ALL_STREAMS_RULES`
- `DBA_STREAMS_RULES`

In addition, these two views display the current **rule condition** for each rule and whether the rule condition has been modified.

The query in this section displays the following information about all of the rules used by Streams clients in a database:

- The name of each Streams client that uses the rule
- The type of each Streams client that uses the rule, either `CAPTURE` for capture process, `PROPAGATION` for propagation, `APPLY` for apply process, or `DEQUEUE` for messaging client
- The name of the rule
- The type of rule set that contains the rule for the Streams client, either `POSITIVE` or `NEGATIVE`
- For Streams rules, the Streams rule level, either `GLOBAL`, `SCHEMA`, or `TABLE`
- For Streams rules, the name of the schema for **schema rule**s and **table rule**s
- For Streams rules, the name of the table for table rules
- For Streams rules, the rule type, either `DML` or `DDL`

Run the following query to display this information:

```
COLUMN STREAMS_NAME HEADING 'Streams|Name' FORMAT A14
COLUMN STREAMS_TYPE HEADING 'Streams|Type' FORMAT A11
COLUMN RULE_NAME HEADING 'Rule|Name' FORMAT A12
COLUMN RULE_SET_TYPE HEADING 'Rule Set|Type' FORMAT A8
COLUMN STREAMS_RULE_TYPE HEADING 'Streams|Rule|Level' FORMAT A7
COLUMN SCHEMA_NAME HEADING 'Schema|Name' FORMAT A6
COLUMN OBJECT_NAME HEADING 'Object|Name' FORMAT A11
COLUMN RULE_TYPE HEADING 'Rule|Type' FORMAT A4

SELECT STREAMS_NAME,
       STREAMS_TYPE,
       RULE_NAME,
       RULE_SET_TYPE,
       STREAMS_RULE_TYPE,
       SCHEMA_NAME,
       OBJECT_NAME,
       RULE_TYPE
  FROM DBA_STREAMS_RULES;
```

Your output looks similar to the following:

```
                                                 Streams
Streams         Streams     Rule          Rule Set Rule    Schema Object      Rule
Name            Type        Name          Type     Level   Name   Name        Type
--------------- ----------- ------------ -------- ------- ------ ----------- ----
STRM01_CAPTURE CAPTURE     JOBS4         POSITIVE TABLE   HR     JOBS        DML
STRM01_CAPTURE CAPTURE     JOBS5         POSITIVE TABLE   HR     JOBS        DDL
DBS1_TO_DBS2   PROPAGATION HR18          POSITIVE SCHEMA  HR                 DDL
DBS1_TO_DBS2   PROPAGATION HR17          POSITIVE SCHEMA  HR                 DML
APPLY          APPLY       HR20          POSITIVE SCHEMA  HR                 DML
APPLY          APPLY       JOB_HISTORY2 NEGATIVE TABLE   HR     JOB_HISTORY DML
OE             DEQUEUE     RULE$_28      POSITIVE
```

This output provides the following information about the rules used by Streams clients in the database:

- The DML rule `jobs4` and the DDL rule `jobs5` are both table rules for the `hr.jobs` table in the positive rule set for the capture process `strm01_capture`.
- The DML rule `hr17` and the DDL rule `hr18` are both **schema rule**s for the `hr` schema in the positive rule set for the propagation `dbs1_to_dbs2`.
- The DML rule `hr20` is a schema rule for the `hr` schema in the positive rule set for the apply process `apply`.
- The DML rule `job_history2` is a table rule for the `hr` schema in the negative rule set for the apply process `apply`.
- The rule `rule$_28` is a messaging rule in the positive rule set for the messaging client `oe`.

The `ALL_STREAMS_RULES` and `DBA_STREAMS_RULES` views also contain information about the rule sets used by a Streams client, the current and original rule condition for Streams rules, whether the rule condition has been changed, the subsetting operation and DML condition for each Streams **subset rule**, the **source database** specified for each Streams rule, and information about the **message** type and message variable for Streams messaging rules.

The following data dictionary views also display Streams rules:

- ALL_STREAMS_GLOBAL_RULES
- DBA_STREAMS_GLOBAL_RULES
- ALL_STREAMS_MESSAGE_RULES
- DBA_STREAMS_MESSAGE_RULES
- ALL_STREAMS_SCHEMA_RULES
- DBA_STREAMS_SCHEMA_RULES
- ALL_STREAMS_TABLE_RULES
- DBA_STREAMS_TABLE_RULES

These views display Streams rules only. They do not display any manual modifications to these rules made by the DBMS_RULE_ADM package, and they do not display rules created using the DBMS_RULE_ADM package. These views can display the original rule condition for each rule only. They do not display the current rule condition for a rule if the rule condition was modified after the rule was created.

Displaying the Streams Rules Used by a Specific Streams Client

To determine which **rule**s are in a **rule set** used by a particular **Streams client**, you can query the DBA_STREAMS_RULES data dictionary view. For example, suppose a database is running an **apply process** named strm01_apply. The following sections describe how to determine the rules in the **positive rule set** and **negative rule set** for this apply process.

The following sections describe how to determine which rules are in a rule set used by a particular Streams client:

- Displaying the Rules in the Positive Rule Set for a Streams Client
- Displaying the Rules in the Negative Rule Set for a Streams Client

See Also: "System-Created Rules" on page 6-5

Displaying the Rules in the Positive Rule Set for a Streams Client

The following query displays all of the **rule**s in the **positive rule set** for an **apply process**s named strm01_apply:

```
COLUMN RULE_OWNER HEADING 'Rule Owner' FORMAT A10
COLUMN RULE_NAME HEADING 'Rule|Name' FORMAT A12
COLUMN STREAMS_RULE_TYPE HEADING 'Streams|Rule|Level' FORMAT A7
COLUMN SCHEMA_NAME HEADING 'Schema|Name' FORMAT A6
COLUMN OBJECT_NAME HEADING 'Object|Name' FORMAT A11
COLUMN RULE_TYPE HEADING 'Rule|Type' FORMAT A4
COLUMN SOURCE_DATABASE HEADING 'Source' FORMAT A10
COLUMN INCLUDE_TAGGED_LCR HEADING 'Apply|Tagged|LCRs?' FORMAT A9
```

```
SELECT RULE_OWNER,
       RULE_NAME,
       STREAMS_RULE_TYPE,
       SCHEMA_NAME,
       OBJECT_NAME,
       RULE_TYPE,
       SOURCE_DATABASE,
       INCLUDE_TAGGED_LCR
  FROM DBA_STREAMS_RULES
  WHERE STREAMS_NAME  = 'STRM01_APPLY' AND
        RULE_SET_TYPE = 'POSITIVE';
```

If this query returns any rows, then the apply process applies LCRs containing changes that evaluate to `TRUE` for the rules.

Your output looks similar to the following:

```
                         Streams                                  Apply
           Rule          Rule    Schema Object      Rule         Tagged
Rule Owner Name          Level   Name   Name        Type Source   LCRs?
---------- ------------- ------- ------ ----------- ---- ------- ---------
STRMADMIN  HR20          SCHEMA  HR                 DML  DBS1.NET NO
STRMADMIN  HR21          SCHEMA  HR                 DDL  DBS1.NET NO
```

Assuming the **rule condition**s for the Streams rules returned by this query have not been modified, these results show that the apply process applies LCRs containing DML changes and DDL changes to the `hr` schema and that the LCRs originated at the `dbs1.net` database. The rules in the positive rule set that instruct the apply process to apply these LCRs are owned by the `strmadmin` user and are named `hr20` and `hr21`. Also, the apply process applies an LCR that satisfies one of these rules only if the **tag** in the LCR is `NULL`.

If the rule condition for a Streams rule has been modified, then you must check the current rule condition to determine the effect of the rule on a **Streams client**. Streams rules whose rule condition has been modified have `NO` for the `SAME_RULE_CONDITION` column.

> **See Also:** "Displaying Modified Rule Conditions for Streams Rules" on page 23-7

Displaying the Rules in the Negative Rule Set for a Streams Client

The following query displays all of the **rule**s in the **negative rule set** for an **apply process** named `strm01_apply`:

```
COLUMN RULE_OWNER HEADING 'Rule Owner' FORMAT A10
COLUMN RULE_NAME HEADING 'Rule|Name' FORMAT A15
COLUMN STREAMS_RULE_TYPE HEADING 'Streams|Rule|Level' FORMAT A7
COLUMN SCHEMA_NAME HEADING 'Schema|Name' FORMAT A6
COLUMN OBJECT_NAME HEADING 'Object|Name' FORMAT A11
COLUMN RULE_TYPE HEADING 'Rule|Type' FORMAT A4
COLUMN SOURCE_DATABASE HEADING 'Source' FORMAT A10
COLUMN INCLUDE_TAGGED_LCR HEADING 'Apply|Tagged|LCRs?' FORMAT A9
```

```
SELECT RULE_OWNER,
       RULE_NAME,
       STREAMS_RULE_TYPE,
       SCHEMA_NAME,
       OBJECT_NAME,
       RULE_TYPE,
       SOURCE_DATABASE,
       INCLUDE_TAGGED_LCR
  FROM DBA_STREAMS_RULES
  WHERE STREAMS_NAME  = 'APPLY' AND
        RULE_SET_TYPE = 'NEGATIVE';
```

If this query returns any rows, then the apply process discards LCRs containing changes that evaluate to `TRUE` for the rules.

Your output looks similar to the following:

```
                         Streams                                    Apply
           Rule          Rule    Schema Object      Rule            Tagged
Rule Owner Name          Level   Name   Name        Type Source     LCRs?
---------- ------------- ------- ------ ----------- ---- ---------- ---------
STRMADMIN  JOB_HISTORY22 TABLE   HR     JOB_HISTORY DML  DBS1.NET   YES
STRMADMIN  JOB_HISTORY23 TABLE   HR     JOB_HISTORY DDL  DBS1.NET   YES
```

Assuming the **rule condition**s for the Streams rules returned by this query have not been modified, these results show that the apply process discards LCRs containing DML changes and DDL changes to the `hr.job_history` table and that the LCRs originated at the `dbs1.net` database. The rules in the negative rule set that instruct the apply process to discard these LCRs are owned by the `strmadmin` user and are named `job_history22` and `job_history23`. Also, the apply process discards an LCR that satisfies one of these rules regardless of the value of the **tag** in the LCR.

If the rule condition for a Streams rule has been modified, then you must check the current rule condition to determine the effect of the rule on a **Streams client**. Streams rules whose rule condition has been modified have `NO` for the `SAME_RULE_CONDITION` column.

> **See Also:** "Displaying Modified Rule Conditions for Streams Rules" on page 23-7

Displaying the Current Condition for a Rule

If you know the name of a **rule**, then you can display its **rule condition**. For example, consider the rule returned by the query in "Displaying the Streams Rules Used by a Specific Streams Client" on page 23-4. The name of the rule is `hr1`, and you can display its condition by running the following query:

```
SET LONG  8000
SET PAGES 8000
SELECT RULE_CONDITION "Current Rule Condition"
  FROM DBA_STREAMS_RULES
  WHERE RULE_NAME  = 'HR1' AND
        RULE_OWNER = 'STRMADMIN';
```

Your output looks similar to the following:

```
Current Rule Condition
--------------------------------------------------------------------
(:dml.get_object_owner() = 'HR' and :dml.is_null_tag() = 'Y' and
:dml.get_source_database_name() = 'DBS1.NET' )
```

See Also:

- "Rule Condition" on page 5-2
- "System-Created Rules" on page 6-5

Displaying Modified Rule Conditions for Streams Rules

It is possible to modify the **rule condition** of a Streams **rule**. These modifications can change the behavior of the **Streams client**s using the Streams rule. In addition, some modifications can degrade rule evaluation performance.

The following query displays the rule name, the original rule condition, and the current rule condition for each Streams rule whose condition has been modified:

```
COLUMN RULE_NAME HEADING 'Rule Name' FORMAT A12
COLUMN ORIGINAL_RULE_CONDITION HEADING 'Original Rule Condition' FORMAT A33
COLUMN RULE_CONDITION HEADING 'Current Rule Condition' FORMAT A33

SET LONG  8000
SET PAGES 8000
SELECT RULE_NAME, ORIGINAL_RULE_CONDITION, RULE_CONDITION
  FROM DBA_STREAMS_RULES
  WHERE SAME_RULE_CONDITION = 'NO';
```

Your output looks similar to the following:

```
Rule Name    Original Rule Condition           Current Rule Condition
------------ --------------------------------- ---------------------------------
HR20         ((:dml.get_object_owner() = 'HR') ((:dml.get_object_owner() = 'HR')
              and :dml.is_null_tag() = 'Y' )    and :dml.is_null_tag() = 'Y' and
                                                :dml.get_object_name() != 'JOB_H
                                               ISTORY')
```

In this example, the output shows that the condition of the `hr20` rule has been modified. Originally, this **schema rule** evaluated to `TRUE` for all changes to the `hr` schema. The current modified condition for this rule evaluates to `TRUE` for all changes to the `hr` schema, except for DML changes to the `hr.job_history` table.

Note: The query in this section applies only to Streams rules. It does not apply to rules created using the `DBMS_RULE_ADM` package because these rules always show `NULL` for the `ORIGINAL_RULE_CONDITION` column and `NULL` for the `SAME_RULE_CONDITION` column.

See Also:

- "Rule Condition" on page 5-2
- "System-Created Rules" on page 6-5

Displaying the Evaluation Context for Each Rule Set

The following query displays the default **evaluation context** for each **rule set** in a database:

```
COLUMN RULE_SET_OWNER HEADING 'Rule Set|Owner' FORMAT A10
COLUMN RULE_SET_NAME HEADING 'Rule Set Name' FORMAT A20
COLUMN RULE_SET_EVAL_CONTEXT_OWNER HEADING 'Eval Context|Owner' FORMAT A12
COLUMN RULE_SET_EVAL_CONTEXT_NAME HEADING 'Eval Context Name' FORMAT A30

SELECT RULE_SET_OWNER,
       RULE_SET_NAME,
       RULE_SET_EVAL_CONTEXT_OWNER,
       RULE_SET_EVAL_CONTEXT_NAME
  FROM DBA_RULE_SETS;
```

Your output looks similar to the following:

```
Rule Set                       Eval Context
Owner      Rule Set Name       Owner        Eval Context Name
---------- ------------------- ------------ ------------------------------
STRMADMIN  RULESET$_2          SYS          STREAMS$_EVALUATION_CONTEXT
STRMADMIN  STRM02_QUEUE_R      STRMADMIN    AQ$_STRM02_QUEUE_TABLE_V
STRMADMIN  APPLY_OE_RS         STRMADMIN    OE_EVAL_CONTEXT
STRMADMIN  OE_QUEUE_R          STRMADMIN    AQ$_OE_QUEUE_TABLE_V
STRMADMIN  AQ$_1_RE            STRMADMIN    AQ$_OE_QUEUE_TABLE_V
SUPPORT    RS                  SUPPORT      EVALCTX
OE         NOTIFICATION_QUEUE_R OE          AQ$_NOTIFICATION_QUEUE_TABLE_V
```

See Also:

- "Rule Evaluation Context" on page 5-5
- "Evaluation Contexts Used in Streams" on page 6-34

Displaying Information About the Tables Used by an Evaluation Context

The following query displays information about the tables used by an **evaluation context** named `evalctx`, which is owned by the `support` user:

```
COLUMN TABLE_ALIAS HEADING 'Table Alias' FORMAT A20
COLUMN TABLE_NAME HEADING 'Table Name' FORMAT A40

SELECT TABLE_ALIAS,
       TABLE_NAME
  FROM DBA_EVALUATION_CONTEXT_TABLES
  WHERE EVALUATION_CONTEXT_OWNER = 'SUPPORT' AND
        EVALUATION_CONTEXT_NAME = 'EVALCTX';
```

Your output looks similar to the following:

```
Table Alias          Table Name
-------------------- ----------------------------------------
PROB                 problems
```

See Also: "Rule Evaluation Context" on page 5-5

Displaying Information About the Variables Used in an Evaluation Context

The following query displays information about the variables used by an **evaluation context** named `evalctx`, which is owned by the `support` user:

```
COLUMN VARIABLE_NAME HEADING 'Variable Name' FORMAT A15
COLUMN VARIABLE_TYPE HEADING 'Variable Type' FORMAT A15
COLUMN VARIABLE_VALUE_FUNCTION HEADING 'Variable Value|Function' FORMAT A20
COLUMN VARIABLE_METHOD_FUNCTION HEADING 'Variable Method|Function' FORMAT A20

SELECT VARIABLE_NAME,
       VARIABLE_TYPE,
       VARIABLE_VALUE_FUNCTION,
       VARIABLE_METHOD_FUNCTION
  FROM DBA_EVALUATION_CONTEXT_VARS
  WHERE EVALUATION_CONTEXT_OWNER = 'SUPPORT' AND
        EVALUATION_CONTEXT_NAME = 'EVALCTX';
```

Your output looks similar to the following:

```
                               Variable Value       Variable Method
Variable Name   Variable Type  Function             Function
--------------- --------------- -------------------- --------------------
CURRENT_TIME    DATE            timefunc
```

See Also: "Rule Evaluation Context" on page 5-5

Displaying All of the Rules in a Rule Set

The query in this section displays the following information about all of the **rule**s in a **rule set**:

- The owner of the rule.
- The name of the rule.
- The **evaluation context** for the rule, if any. If a rule does not have an evaluation context, and no evaluation context is specified in the `ADD_RULE` procedure when the rule is added to a rule set, then it inherits the evaluation context of the rule set.
- The evaluation context owner, if the rule has an evaluation context.

For example, to display this information for each rule in a rule set named `oe_queue_r` that is owned by the user `strmadmin`, run the following query:

```
COLUMN RULE_OWNER HEADING 'Rule Owner' FORMAT A10
COLUMN RULE_NAME HEADING 'Rule Name' FORMAT A20
COLUMN RULE_EVALUATION_CONTEXT_NAME HEADING 'Eval Context Name' FORMAT A27
COLUMN RULE_EVALUATION_CONTEXT_OWNER HEADING 'Eval Context|Owner' FORMAT A11

SELECT R.RULE_OWNER,
       R.RULE_NAME,
       R.RULE_EVALUATION_CONTEXT_NAME,
       R.RULE_EVALUATION_CONTEXT_OWNER
  FROM DBA_RULES R, DBA_RULE_SET_RULES RS
  WHERE RS.RULE_SET_OWNER = 'STRMADMIN' AND
  RS.RULE_SET_NAME = 'OE_QUEUE_R' AND
  RS.RULE_NAME = R.RULE_NAME AND
  RS.RULE_OWNER = R.RULE_OWNER;
```

Your output looks similar to the following:

```
                                                                Eval Contex
Rule Owner Rule Name            Eval Context Name               Owner
---------- -------------------- ------------------------------ -----------
STRMADMIN  HR1                  STREAMS$_EVALUATION_CONTEXT SYS
STRMADMIN  APPLY_LCRS           STREAMS$_EVALUATION_CONTEXT SYS
STRMADMIN  OE_QUEUE$3
STRMADMIN  APPLY_ACTION
```

Displaying the Condition for Each Rule in a Rule Set

The following query displays the condition for each **rule** in a **rule set** named `hr_queue_r` that is owned by the user `strmadmin`:

```
SET LONGCHUNKSIZE 4000
SET LONG 4000
COLUMN RULE_OWNER HEADING 'Rule Owner' FORMAT A15
COLUMN RULE_NAME HEADING 'Rule Name' FORMAT A15
COLUMN RULE_CONDITION HEADING 'Rule Condition' FORMAT A45

SELECT R.RULE_OWNER,
       R.RULE_NAME,
       R.RULE_CONDITION
  FROM DBA_RULES R, DBA_RULE_SET_RULES RS
  WHERE RS.RULE_SET_OWNER = 'STRMADMIN' AND
  RS.RULE_SET_NAME = 'HR_QUEUE_R' AND
  RS.RULE_NAME = R.RULE_NAME AND
  RS.RULE_OWNER = R.RULE_OWNER;
```

Your output looks similar to the following:

```
Rule Owner      Rule Name       Rule Condition
--------------- --------------- --------------------------------------------
STRMADMIN       APPLY_ACTION     hr.get_hr_action(tab.user_data) = 'APPLY'
STRMADMIN       APPLY_LCRS      :dml.get_object_owner() = 'HR' AND  (:dml.get
                                _object_name() = 'DEPARTMENTS' OR
                                :dml.get_object_name() = 'EMPLOYEES')

STRMADMIN       HR_QUEUE$3      hr.get_hr_action(tab.user_data) != 'APPLY'
```

See Also:

- "Rule Condition" on page 5-2
- "System-Created Rules" on page 6-5

Listing Each Rule that Contains a Specified Pattern in Its Condition

To list each **rule** in a database that contains a specified pattern in its condition, you can query the `DBMS_RULES` data dictionary view and use the `DBMS_LOB.INSTR` function to search for the pattern in the **rule condition**s. For example, the following query lists each rule that contains the pattern `'HR'` in its condition:

```
COLUMN RULE_OWNER HEADING 'Rule Owner' FORMAT A30
COLUMN RULE_NAME HEADING 'Rule Name' FORMAT A30

SELECT RULE_OWNER, RULE_NAME FROM DBA_RULES
  WHERE DBMS_LOB.INSTR(RULE_CONDITION, 'HR', 1, 1) > 0;
```

Your output looks similar to the following:

```
Rule Owner                     Rule Name
------------------------------ ------------------------------
STRMADMIN                      DEPARTMENTS4
STRMADMIN                      DEPARTMENTS5
STRMADMIN                      DEPARTMENTS6
```

Displaying Aggregate Statistics for All Rule Set Evaluations

You can query the V$RULE_SET_AGGREGATE_STATS dynamic performance view to display statistics for all **rule set** evaluations since the database instance last started.

The query in this section contains the following information about rule set evaluations:

- The number of rule set evaluations.
- The number of rule set evaluations that were instructed to stop on the first hit.
- The number of rule set evaluations that were instructed to evaluate only simple **rule**s.
- The number of times a rule set was evaluated without issuing any SQL. Generally, issuing SQL to evaluate rules is more expensive than evaluating rules without issuing SQL.
- The number of centiseconds of CPU time used for rule set evaluation.
- The number of centiseconds spent on rule set evaluation.
- The number of SQL executions issued to evaluate a rule in a rule set.
- The number of **rule condition**s processed during rule set evaluation.
- The number of TRUE rules returned to the **rules engine** clients.
- The number of MAYBE rules returned to the rules engine clients.
- The number of times the following types of functions were called during rule set evaluation: variable value function, variable method function, and evaluation function.

Run the following query to display this information:

```
COLUMN NAME HEADING 'Name of Statistic' FORMAT A55
COLUMN VALUE HEADING 'Value' FORMAT 999999999

SELECT NAME, VALUE FROM V$RULE_SET_AGGREGATE_STATS;
```

Your output looks similar to the following:

```
Name of Statistic                                       Value
------------------------------------------------------- ----------
rule set evaluations (all)                                5584
rule set evaluations (first_hit)                          5584
rule set evaluations (simple_rules_only)                  3675
rule set evaluations (SQL free)                           5584
rule set evaluation time (CPU)                             179
rule set evaluation time (elapsed)                        1053
rule set SQL executions                                      0
rule set conditions processed                            11551
rule set true rules                                         10
rule set maybe rules                                       328
```

```
rule set user function calls (variable value function)         182
rule set user function calls (variable method function)      12794
rule set user function calls (evaluation function)            3857
```

Note: A centisecond is one-hundredth of a second. So, for example, this output shows 1.79 seconds of CPU time and 10.53 seconds of elapsed time.

Displaying Information About Evaluations for Each Rule Set

You can query the V$RULE_SET dynamic performance view to display information about evaluations for each **rule set** since the database instance last started. The query in this section contains the following information about each rule set in a database:

- The owner of the rule set.
- The name of the rule set.
- The total number of evaluations of the rule set since the database instance last started.
- The total number of times SQL was executed to evaluate **rule**s since the database instance last started. Generally, issuing SQL to evaluate rules is more expensive than evaluating rules without issuing SQL.
- The total number of evaluations on the rule set that did not issue SQL to evaluate rules since the database instance last started.
- The total number of TRUE rules returned to the **rules engine** clients using the rule set since the database instance last started.
- The total number of MAYBE rules returned to the rules engine clients using the rule set since the database instance last started.

Run the following query to display this information for each rule set in the database:

```
COLUMN OWNER HEADING 'Rule Set|Owner' FORMAT A9
COLUMN NAME HEADING 'Rule Set|Name' FORMAT A11
COLUMN EVALUATIONS HEADING 'Total|Evaluations' FORMAT 999999
COLUMN SQL_EXECUTIONS HEADING 'SQL|Executions' FORMAT 999999
COLUMN SQL_FREE_EVALUATIONS HEADING 'SQL Free|Evaluations' FORMAT 999999
COLUMN TRUE_RULES HEADING 'True|Rules' FORMAT 999999
COLUMN MAYBE_RULES HEADING 'Maybe|Rules' FORMAT 999999

SELECT OWNER,
       NAME,
       EVALUATIONS,
       SQL_EXECUTIONS,
       SQL_FREE_EVALUATIONS,
       TRUE_RULES,
       MAYBE_RULES
  FROM V$RULE_SET;
```

Your output looks similar to the following:

```
Rule Set  Rule Set          Total        SQL    SQL Free    True   Maybe
Owner     Name        Evaluations Executions Evaluations   Rules   Rules
--------- ----------- ----------- ---------- ----------- ------- -------
STRMADMIN RULESET$_18         403          0         403       0     200
STRMADMIN RULESET$_9         3454          0        3454       5      64
```

Note: Querying the V$RULE_SET view can have a negative impact on performance if a database has a large library cache.

Determining the Resources Used by Evaluation of Each Rule Set

You can query the V$RULE_SET dynamic performance view to determine the resources used by evaluation of a **rule set** since the database instance last started. If a rule set was evaluated more than one time since the database instance last started, then some statistics are cumulative, including statistics for the amount of CPU time, evaluation time, and shared memory bytes used.

The query in this section contains the following information about each rule set in a database:

- The owner of the rule set
- The name of the rule set
- The total number of seconds of CPU time used to evaluate the rule set since the database instance last started
- The total number of seconds used to evaluate the rule set since the database instance last started
- The total number of shared memory bytes used to evaluate the rule set since the database instance last started

Run the following query to display this information for each rule set in the database:

```
COLUMN OWNER HEADING 'Rule Set|Owner' FORMAT A15
COLUMN NAME HEADING 'Rule Set Name' FORMAT A15
COLUMN CPU_SECONDS HEADING 'Seconds|of CPU|Time' FORMAT 999999.999
COLUMN ELAPSED_SECONDS HEADING 'Seconds of|Evaluation|Time' FORMAT 999999.999
COLUMN SHARABLE_MEM HEADING 'Bytes|of Shared|Memory' FORMAT 999999999

SELECT OWNER,
       NAME,
       (CPU_TIME/100) CPU_SECONDS,
       (ELAPSED_TIME/100) ELAPSED_SECONDS,
       SHARABLE_MEM
  FROM V$RULE_SET;
```

Your output looks similar to the following:

```
                                    Seconds  Seconds of      Bytes
Rule Set                             of CPU  Evaluation  of Shared
Owner           Rule Set Name          Time        Time     Memory
--------------- --------------- ----------- ----------- ----------
STRMADMIN       RULESET$_18            .840       8.550     444497
STRMADMIN       RULESET$_9             .700       1.750     444496
```

Note: Querying the V$RULE_SET view can have a negative impact on performance if a database has a large library cache.

Displaying Evaluation Statistics for a Rule

You can query the V$RULE dynamic performance view to display evaluation statistics for a particular **rule** since the database instance last started. The query in this section contains the following information about each **rule set** in a database:

- The total number of times the rule evaluated to TRUE since the database instance last started.
- The total number of times the rule evaluated to MAYBE since the database instance last started.
- The total number of evaluations on the rule that issued SQL since the database instance last started. Generally, issuing SQL to evaluate a rule is more expensive than evaluating the rule without issuing SQL.

For example, run the following query to display this information for the locations25 rule in the strmadmin schema:

```
COLUMN TRUE_HITS HEADING 'True Evaluations' FORMAT 999999
COLUMN MAYBE_HITS HEADING 'Maybe Evaluations' FORMAT 999999
COLUMN SQL_EVALUATIONS HEADING 'SQL Evaluations' FORMAT 999999

SELECT TRUE_HITS, MAYBE_HITS, SQL_EVALUATIONS
  FROM V$RULE
  WHERE RULE_OWNER = 'STRMADMIN' AND
        RULE_NAME  = 'LOCATIONS25';
```

Your output looks similar to the following:

```
True Evaluations Maybe Evaluations SQL Evaluations
---------------- ----------------- ---------------
            1518               154               0
```

24

Monitoring Rule-Based Transformations

A rule-based transformation is any modification to a **message** that results when a **rule** in a **positive rule set** evaluates to `TRUE`. This chapter provides sample queries that you can use to monitor rule-based transformations.

This chapter contains these topics:

- Displaying Information About All Rule-Based Transformations
- Displaying Declarative Rule-Based Transformations
- Displaying Custom Rule-Based Transformations

Note: The Streams tool in the Oracle Enterprise Manager Console is also an excellent way to monitor a Streams environment. See the online help for the Streams tool for more information.

See Also:

- Chapter 7, "Rule-Based Transformations"
- Chapter 15, "Managing Rule-Based Transformations"
- *Oracle Database Reference* for information about the data dictionary views described in this chapter
- *Oracle Streams Replication Administrator's Guide* for information about monitoring a Streams **replication** environment

Displaying Information About All Rule-Based Transformations

The query in this section displays the following information about each **rule-based transformation** in a database:

- The owner of the **rule** for which a rule-based transformation is specified
- The name of the rule for which a rule-based transformation is specified
- The type of rule-based transformation:
  - `SUBSET RULE` is displayed for **subset rule**s, which use internal rule-based transformations.
  - `DECLARATIVE TRANSFORMATION` is displayed for **declarative rule-based transformation**s.
  - `CUSTOM TRANSFORMATION` is displayed for **custom rule-based transformation**s.

Run the following query to display this information for the rule-based transformations in a database:

```
COLUMN RULE_OWNER HEADING 'Rule Owner' FORMAT A20
COLUMN RULE_NAME HEADING 'Rule Name' FORMAT A20
COLUMN TRANSFORM_TYPE HEADING 'Transformation Type' FORMAT A30

SELECT RULE_OWNER,
       RULE_NAME,
       TRANSFORM_TYPE
  FROM DBA_STREAMS_TRANSFORMATIONS;
```

Your output looks similar to the following:

```
Rule Owner           Rule Name            Transformation Type
-------------------- -------------------- ------------------------------
STRMADMIN            EMPLOYEES23          DECLARATIVE TRANSFORMATION
STRMADMIN            JOBS26               DECLARATIVE TRANSFORMATION
STRMADMIN            DEPARTMENTS33        SUBSET RULE
STRMADMIN            DEPARTMENTS32        SUBSET RULE
STRMADMIN            DEPARTMENTS34        SUBSET RULE
STRMADMIN            DEPARTMENTS32        CUSTOM TRANSFORMATION
STRMADMIN            DEPARTMENTS33        CUSTOM TRANSFORMATION
STRMADMIN            DEPARTMENTS34        CUSTOM TRANSFORMATION
```

Displaying Declarative Rule-Based Transformations

A **declarative rule-based transformation** is a rule-based transformation that covers one of a common set of transformation scenarios for row LCRs. Declarative rule-based transformations are run internally without using PL/SQL.

The query in this section displays the following information about each declarative rule-based transformation in a database:

- The owner of the **rule** for which a declarative rule-based transformation is specified.
- The name of the rule for which a declarative rule-based transformation is specified.
- The type of declarative rule-based transformation specified. The following types are possible: `ADD COLUMN`, `DELETE COLUMN`, `RENAME COLUMN`, `RENAME SCHEMA`, and `RENAME TABLE`.
- The precedence of the declarative rule-based transformation. The precedence is the execution order of a transformation in relation to other transformations with the same step number specified for the same rule. For transformations with the same step number, the transformation with the lowest precedence is executed first.
- The step number of the declarative rule-based transformation. If more than one declarative rule-based transformation is specified for the same rule, then the transformation with the lowest step number is executed first. You can specify the step number for a declarative rule-based transformation when you create the transformation.

Run the following query to display this information for the declarative rule-based transformations in a database:

```
COLUMN RULE_OWNER HEADING 'Rule Owner' FORMAT A15
COLUMN RULE_NAME HEADING 'Rule Name' FORMAT A15
COLUMN DECLARATIVE_TYPE HEADING 'Declarative|Type' FORMAT A15
COLUMN PRECEDENCE HEADING 'Precedence' FORMAT 99999
COLUMN STEP_NUMBER HEADING 'Step Number' FORMAT 99999

SELECT RULE_OWNER,
       RULE_NAME,
       DECLARATIVE_TYPE,
       PRECEDENCE,
       STEP_NUMBER
  FROM DBA_STREAMS_TRANSFORMATIONS
  WHERE TRANSFORM_TYPE = 'DECLARATIVE TRANSFORMATION';
```

Your output looks similar to the following:

```
                               Declarative
Rule Owner      Rule Name      Type            Precedence Step Number
--------------- -------------- --------------- ---------- -----------
STRMADMIN       JOBS26         RENAME TABLE             4           0
STRMADMIN       EMPLOYEES23    ADD COLUMN               3           0
```

Based on this output, the `ADD COLUMN` transformation executes before the `RENAME TABLE` transformation because the step number is the same (zero) for both transformations and the `ADD COLUMN` transformation has the lower precedence.

When you determine which types of declarative rule-based transformations are in a database, you can display more detailed information about each transformation. The following data dictionary views contain detailed information about the various types of declarative rule-based transformations:

- The `DBA_STREAMS_ADD_COLUMN` view contains information about `ADD COLUMN` declarative transformations.
- The `DBA_STREAMS_DELETE_COLUMN` view contains information about `DELETE COLUMN` declarative transformations.
- The `DBA_STREAMS_RENAME_COLUMN` view contains information about `RENAME COLUMN` declarative transformations.
- The `DBA_STREAMS_RENAME_SCHEMA` view contains information about `RENAME SCHEMA` declarative transformations.
- The `DBA_STREAMS_RENAME_TABLE` view contains information about `RENAME TABLE` declarative transformations.

For example, the previous query listed an `ADD COLUMN` transformation and a `RENAME TABLE` transformation. The following sections contain queries that display detailed information about these transformations:

- Displaying Information About ADD COLUMN Transformations
- Displaying Information About RENAME TABLE Transformations

> **Note:** Precedence and step number pertain only to declarative rule-based transformations. They do not pertain to **subset rule** transformations or **custom rule-based transformation**s.

See Also:

- "Declarative Rule-Based Transformations" on page 7-1
- "Managing Declarative Rule-Based Transformations" on page 15-1

Displaying Information About ADD COLUMN Transformations

The following query displays detailed information about the ADD COLUMN **declarative rule-based transformation**s in a database:

```
COLUMN RULE_OWNER HEADING 'Rule|Owner' FORMAT A9
COLUMN RULE_NAME HEADING 'Rule|Name' FORMAT A12
COLUMN SCHEMA_NAME HEADING 'Schema|Name' FORMAT A6
COLUMN TABLE_NAME HEADING 'Table|Name' FORMAT A9
COLUMN COLUMN_NAME HEADING 'Column|Name' FORMAT A10
COLUMN COLUMN_TYPE HEADING 'Column|Type' FORMAT A8

SELECT RULE_OWNER,
       RULE_NAME,
       SCHEMA_NAME,
       TABLE_NAME,
       COLUMN_NAME,
       ANYDATA.AccessDate(COLUMN_VALUE) "Value",
       COLUMN_TYPE
  FROM DBA_STREAMS_ADD_COLUMN;
```

Your output looks similar to the following:

```
Rule      Rule         Schema Table     Column                          Column
Owner     Name         Name   Name      Name       Value                Type
--------- ------------ ------ --------- ---------- -------------------- --------
STRMADMIN EMPLOYEES23  HR     EMPLOYEES BIRTH_DATE                       SYS.DATE
```

This output show the following information about the ADD COLUMN declarative rule-based transformation:

- It is specified on the employees23 **rule** in the strmadmin schema.
- It adds a column to row LCRs that involve the employees table in the hr schema.
- The column name of the added column is birth_date.
- The value of the added column is NULL. Notice that the COLUMN_VALUE column in the DBA_STREAMS_ADD_COLUMN view is type ANYDATA. In this example, because the column type is DATE, the ANYDATA.AccessDate member function is used to display the value. Use the appropriate member function to display values of other types.
- The type of the added column is DATE.

Displaying Information About RENAME TABLE Transformations

The following query displays detailed information about the RENAME TABLE **declarative rule-based transformation**s in a database:

```
COLUMN RULE_OWNER HEADING 'Rule|Owner' FORMAT A10
COLUMN RULE_NAME HEADING 'Rule|Name' FORMAT A10
COLUMN FROM_SCHEMA_NAME HEADING 'From|Schema|Name' FORMAT A10
COLUMN TO_SCHEMA_NAME HEADING 'To|Schema|Name' FORMAT A10
COLUMN FROM_TABLE_NAME HEADING 'From|Table|Name' FORMAT A15
COLUMN TO_TABLE_NAME HEADING 'To|Table|Name' FORMAT A15
```

```
SELECT RULE_OWNER,
       RULE_NAME,
       FROM_SCHEMA_NAME,
       TO_SCHEMA_NAME,
       FROM_TABLE_NAME,
       TO_TABLE_NAME
  FROM DBA_STREAMS_RENAME_TABLE;
```

Your output looks similar to the following:

```
                     From       To         From            To
Rule       Rule      Schema     Schema     Table           Table
Owner      Name      Name       Name       Name            Name
---------- ---------- ---------- ---------- --------------- ---------------
STRMADMIN  JOBS26     HR         HR         JOBS            ASSIGNMENTS
```

This output show the following information about the `RENAME TABLE` declarative rule-based transformation:

- It is specified on the `jobs26` **rule** in the `strmadmin` schema.
- It renames the `hr.jobs` table in row LCRs to the `hr.assignments` table.

Displaying Custom Rule-Based Transformations

A **custom rule-based transformation** is a rule-based transformation that requires a user-defined PL/SQL function. The query in this section displays the following information about each custom rule-based transformation specified in a database:

- The owner of the **rule** on which the custom rule-based transformation is set
- The name of the rule on which the custom rule-based transformation is set
- The owner and name of the transformation function
- Whether the custom rule-based transformation is one-to-one or one-to-many

Run the following query to display this information:

```
COLUMN RULE_OWNER HEADING 'Rule Owner' FORMAT A20
COLUMN RULE_NAME HEADING 'Rule Name' FORMAT A15
COLUMN TRANSFORM_FUNCTION_NAME HEADING 'Transformation Function' FORMAT A30
COLUMN CUSTOM_TYPE HEADING 'Type' FORMAT A11

SELECT RULE_OWNER, RULE_NAME, TRANSFORM_FUNCTION_NAME, CUSTOM_TYPE
  FROM DBA_STREAMS_TRANSFORM_FUNCTION;
```

Your output looks similar to the following:

```
Rule Owner           Rule Name       Transformation Function        Type
-------------------- --------------- ------------------------------ -----------
STRMADMIN            DEPARTMENTS31   "HR"."EXECUTIVE_TO_MANAGEMENT" ONE TO ONE
STRMADMIN            DEPARTMENTS32   "HR"."EXECUTIVE_TO_MANAGEMENT" ONE TO ONE
STRMADMIN            DEPARTMENTS33   "HR"."EXECUTIVE_TO_MANAGEMENT" ONE TO ONE
```

Note: The transformation function name must be of type `VARCHAR2`. If it is not, then the value of `TRANSFORM_FUNCTION_NAME` is `NULL`. The `VALUE_TYPE` column in the `DBA_STREAMS_TRANSFORM_FUNCTION` view displays the type of the transform function name.

See Also:

- "Custom Rule-Based Transformations" on page 7-2
- "Managing Custom Rule-Based Transformations" on page 15-5

25

Monitoring File Group and Tablespace Repositories

A **file group repository** can contain multiple **file group**s and multiple versions of a particular file group. A **tablespace repository** is a collection of tablespace sets in a file group repository. Tablespace repositories are built on file group repositories, but tablespace repositories only contain the files required to move or copy tablespaces between databases. This chapter provides sample queries that you can use to monitor file group repositories and tablespace repositories.

This chapter contains these topics:

- Monitoring a File Group Repository
- Monitoring a Tablespace Repository

Note: The Streams tool in the Oracle Enterprise Manager Console is also an excellent way to monitor a Streams environment. See the online help for the Streams tool for more information.

See Also:

- Chapter 8, "Information Provisioning"
- Chapter 16, "Using Information Provisioning"
- *Oracle Database Reference* for information about the data dictionary views described in this chapter
- *Oracle Streams Replication Administrator's Guide* for information about monitoring a Streams **replication** environment

Monitoring a File Group Repository

The queries in the following sections provide examples for monitoring a **file group repository**:

- Displaying General Information About the File Groups in a Database
- Displaying Information About File Group Versions
- Displaying Information About File Group Files

See Also:

- "File Group Repository" on page 8-4
- "Using a File Group Repository" on page 16-14

Displaying General Information About the File Groups in a Database

The query in this section displays the following information for each **file group** in the local database:

- The file group owner
- The file group name
- Whether the files in a **version** of the file group are kept on disk if the version is purged
- The minimum number of versions of the file group allowed
- The maximum number of versions of the file group allowed
- The number of days to retain a file group version after it is created

Run the following query to display this information for the local database:

```
COLUMN FILE_GROUP_OWNER HEADING 'File Group|Owner' FORMAT A10
COLUMN FILE_GROUP_NAME HEADING 'File Group|Name' FORMAT A10
COLUMN KEEP_FILES HEADING 'Keep|Files?' FORMAT A10
COLUMN MIN_VERSIONS HEADING 'Minimum|Number|of Versions' FORMAT 9999
COLUMN MAX_VERSIONS HEADING 'Maximum|Number|of Versions' FORMAT 9999999999
COLUMN RETENTION_DAYS HEADING 'Days to|Retain|a Version' FORMAT 9999999999.99

SELECT FILE_GROUP_OWNER,
       FILE_GROUP_NAME,
       KEEP_FILES,
       MIN_VERSIONS,
       MAX_VERSIONS,
       RETENTION_DAYS
  FROM DBA_FILE_GROUPS;
```

Your output looks similar to the following:

```
                                      Minimum     Maximum        Days to
File Group File Group Keep             Number      Number         Retain
Owner      Name       Files?     of Versions of Versions      a Version
---------- ---------- ---------- ----------- ----------- --------------
STRMADMIN  REPORTS    Y                    2  4294967295  4294967295.00
```

This output shows that the database has one file group with the following characteristics:

- The file group owner is `strmadmin`.
- The file group name is `reports`.

- The files in a version are kept on disk if a version is purged because the `"Keep Files?"` is `"Y"` for the file group.
- The minimum number of versions allowed is 2. If the file group automatically purges versions, then it will not purge a version if the purge would cause the total number of versions to drop below 2.
- The file group allows an infinite number of versions. The number 4294967295 means an infinite number of versions.
- The file group retains a version of an infinite number of days. The number 4294967295 means an infinite number of days.

Displaying Information About File Group Versions

The query in this section displays the following information for each **file group version** in the local database:

- The owner of the file group that contains the version
- The name of the file group that contains the version
- The version name
- The version number
- The name of the user who created the version
- Comments for the version

Run the following query to display this information for the local database:

```
COLUMN FILE_GROUP_OWNER HEADING 'File Group|Owner' FORMAT A10
COLUMN FILE_GROUP_NAME HEADING 'File Group|Name' FORMAT A10
COLUMN VERSION_NAME HEADING 'Version Name' FORMAT A20
COLUMN VERSION HEADING 'Version|Number' FORMAT 99999999
COLUMN CREATOR HEADING 'Creator' FORMAT A10
COLUMN COMMENTS HEADING 'Comments' FORMAT A14

SELECT FILE_GROUP_OWNER,
       FILE_GROUP_NAME,
       VERSION_NAME,
       VERSION,
       CREATOR,
       COMMENTS
  FROM DBA_FILE_GROUP_VERSIONS;
```

Your output looks similar to the following:

```
File Group File Group                          Version
Owner      Name       Version Name             Number Creator    Comments
---------- ---------- -------------------- --------- ---------- --------------
STRMADMIN  REPORTS    SALES_REPORTS_V1             1 STRMADMIN  Sales reports
                                                                for week of 06
                                                                -FEB-2005

STRMADMIN  REPORTS    SALES_REPORTS_V2             2 STRMADMIN  Sales reports
                                                                for week of 13
                                                                -FEB-2005
```

Displaying Information About File Group Files

The query in this section displays the following information about each file in a **file group version** in the local database:

- The owner of the file group that contains the file
- The name of the file group that contains the file
- The name of the version in the file group that contains the file
- The file name
- The directory object that contains the file

```
COLUMN FILE_GROUP_OWNER HEADING 'File Group|Owner' FORMAT A10
COLUMN FILE_GROUP_NAME HEADING 'File Group|Name' FORMAT A10
COLUMN VERSION_NAME HEADING 'Version Name' FORMAT A20
COLUMN FILE_NAME HEADING 'File Name' FORMAT A15
COLUMN FILE_DIRECTORY HEADING 'File Directory|Object' FORMAT A15

SELECT FILE_GROUP_OWNER,
       FILE_GROUP_NAME,
       VERSION_NAME,
       FILE_NAME,
       FILE_DIRECTORY
  FROM DBA_FILE_GROUP_FILES;
```

Your output looks similar to the following:

```
File Group File Group                                      File Directory
Owner      Name       Version Name         File Name       Object
---------- ---------- -------------------- --------------- ---------------
STRMADMIN  REPORTS    SALES_REPORTS_V1     book_sales.htm  SALES_REPORTS1
STRMADMIN  REPORTS    SALES_REPORTS_V1     music_sales.htm SALES_REPORTS1
STRMADMIN  REPORTS    SALES_REPORTS_V2     book_sales.htm  SALES_REPORTS2
STRMADMIN  REPORTS    SALES_REPORTS_V2     music_sales.htm SALES_REPORTS2
```

Query the `DBA_DIRECTORIES` data dictionary view to determine the corresponding file system directory for a directory object.

Monitoring a Tablespace Repository

The queries in the following sections provide examples for monitoring a **tablespace repository**:

- Displaying Information About the Tablespaces in a Tablespace Repository
- Displaying Information About the Tables in a Tablespace Repository
- Displaying Export Information About Versions in a Tablespace Repository

See Also:

- "Tablespace Repository" on page 8-4
- "Using a Tablespace Repository" on page 16-1

Displaying Information About the Tablespaces in a Tablespace Repository

The query in this section displays the following information about each tablespace in the **tablespace repository** in the local database:

- The owner of the **file group** that contains the tablespace in the tablespace repository
- The name of the file group that contains the tablespace in the tablespace repository
- The name of the **version** that contains the tablespace
- The tablespace name

```
COLUMN FILE_GROUP_OWNER HEADING 'File Group|Owner' FORMAT A15
COLUMN FILE_GROUP_NAME HEADING 'File Group|Name' FORMAT A15
COLUMN VERSION_NAME HEADING 'Version Name' FORMAT A15
COLUMN VERSION HEADING 'Version|Number' FORMAT 99999999
COLUMN TABLESPACE_NAME HEADING 'Tablespace Name' FORMAT A15

SELECT FILE_GROUP_OWNER,
       FILE_GROUP_NAME,
       VERSION_NAME,
       VERSION,
       TABLESPACE_NAME
  FROM DBA_FILE_GROUP_TABLESPACES;
```

Your output looks similar to the following:

```
File Group      File Group                      Version
Owner           Name            Version Name     Number Tablespace Name
--------------- --------------- --------------- --------- ---------------
STRMADMIN       SALES           V_Q1FY2005              1 SALES_TBS1
STRMADMIN       SALES           V_Q1FY2005              1 SALES_TBS2
STRMADMIN       SALES           V_Q2FY2005              3 SALES_TBS1
STRMADMIN       SALES           V_Q2FY2005              3 SALES_TBS2
STRMADMIN       SALES           V_Q1FY2005_R            4 SALES_TBS1
STRMADMIN       SALES           V_Q1FY2005_R            4 SALES_TBS2
STRMADMIN       SALES           V_Q2FY2005_R            5 SALES_TBS1
STRMADMIN       SALES           V_Q2FY2005_R            5 SALES_TBS2
```

Displaying Information About the Tables in a Tablespace Repository

The query in this section displays the following information about each table in the **tablespace repository** in the local database:

- The owner of the **file group** that contains the table in the tablespace repository
- The name of the file group that contains the table in the tablespace repository
- The name of the **version** that contains the table
- The table owner
- The table name
- The tablespace that contains the table

```
COLUMN FILE_GROUP_OWNER HEADING 'File Group|Owner' FORMAT A10
COLUMN FILE_GROUP_NAME HEADING 'File Group|Name' FORMAT A10
COLUMN VERSION_NAME HEADING 'Version Name' FORMAT A15
COLUMN OWNER HEADING 'Table|Owner' FORMAT A10
COLUMN TABLE_NAME HEADING 'Table Name' FORMAT A15
COLUMN TABLESPACE_NAME HEADING 'Tablespace Name' FORMAT A15
```

```
SELECT FILE_GROUP_OWNER,
       FILE_GROUP_NAME,
       VERSION_NAME,
       OWNER,
       TABLE_NAME,
       TABLESPACE_NAME
  FROM DBA_FILE_GROUP_TABLES;
```

Your output looks similar to the following:

```
File Group File Group                Table
Owner      Name       Version Name   Owner      Table Name      Tablespace Name
---------- ---------- -------------- ---------- --------------- ---------------
STRMADMIN  SALES      V_Q1FY2005     SL         ORDERS          SALES_TBS1
STRMADMIN  SALES      V_Q1FY2005     SL         ORDER_ITEMS     SALES_TBS1
STRMADMIN  SALES      V_Q1FY2005     SL         CUSTOMERS       SALES_TBS2
STRMADMIN  SALES      V_Q2FY2005     SL         ORDERS          SALES_TBS1
STRMADMIN  SALES      V_Q2FY2005     SL         ORDER_ITEMS     SALES_TBS1
STRMADMIN  SALES      V_Q2FY2005     SL         CUSTOMERS       SALES_TBS2
STRMADMIN  SALES      V_Q1FY2005_R   SL         ORDERS          SALES_TBS1
STRMADMIN  SALES      V_Q1FY2005_R   SL         ORDER_ITEMS     SALES_TBS1
STRMADMIN  SALES      V_Q1FY2005_R   SL         CUSTOMERS       SALES_TBS2
STRMADMIN  SALES      V_Q2FY2005_R   SL         ORDERS          SALES_TBS1
STRMADMIN  SALES      V_Q2FY2005_R   SL         ORDER_ITEMS     SALES_TBS1
STRMADMIN  SALES      V_Q2FY2005_R   SL         CUSTOMERS       SALES_TBS2
```

Displaying Export Information About Versions in a Tablespace Repository

To display export information about the versions in the **tablespace repository** in the local database, query the `DBA_FILE_GROUP_EXPORT_INFO` data dictionary view. This view only displays information for versions that contain a valid Data Pump export dump file. The query in this section displays the following export information about each **version** in the local database:

- The name of the **file group** that contains the version
- The name of the version
- The export version of the export dump file. The export version corresponds to the version of Data Pump that performed the export.
- The platform on which the export was performed
- The date and time of the export
- The global name of the exporting database

```
COLUMN FILE_GROUP_NAME HEADING 'File Group|Name' FORMAT A10
COLUMN VERSION_NAME HEADING 'Version Name' FORMAT A13
COLUMN EXPORT_VERSION HEADING 'Export|Version' FORMAT A7
COLUMN PLATFORM_NAME HEADING 'Export Platform' FORMAT A17
COLUMN EXPORT_TIME HEADING 'Export Time' FORMAT A17
COLUMN SOURCE_GLOBAL_NAME HEADING 'Export|Database' FORMAT A10

SELECT FILE_GROUP_NAME,
       VERSION_NAME,
       EXPORT_VERSION,
       PLATFORM_NAME,
       TO_CHAR(EXPORT_TIME, 'HH24:MI:SS MM/DD/YY') EXPORT_TIME,
       SOURCE_GLOBAL_NAME
  FROM DBA_FILE_GROUP_EXPORT_INFO;
```

Your output looks similar to the following:

```
File Group               Export                                    Export
Name       Version Name  Version Export Platform   Export Time       Database
---------- ------------- ------- ----------------- ----------------- ----------
SALES      V_Q1FY2005    10.2.0  Linux IA (32-bit) 12:23:52 03/08/05 INST1.NET
SALES      V_Q2FY2005    10.2.0  Linux IA (32-bit) 12:27:37 03/08/05 INST1.NET
SALES      V_Q1FY2005_R  10.2.0  Linux IA (32-bit) 12:39:50 03/08/05 INST2.NET
SALES      V_Q2FY2005_R  10.2.0  Linux IA (32-bit) 12:46:04 03/08/05 INST2.NET
```

26

Monitoring Other Streams Components

This chapter provides sample queries that you can use to monitor various Streams components.

This chapter contains these topics:

- Monitoring Streams Administrators and Other Streams Users
- Monitoring the Streams Pool
- Monitoring Compatibility in a Streams Environment
- Monitoring Streams Performance Using AWR and Statspack

Note: The Streams tool in the Oracle Enterprise Manager Console is also an excellent way to monitor a Streams environment. See the online help for the Streams tool for more information.

See Also:

- *Oracle Database Reference* for information about the data dictionary views described in this chapter
- *Oracle Streams Replication Administrator's Guide* for information about monitoring a Streams **replication** environment

Monitoring Streams Administrators and Other Streams Users

The following sections contain queries that you can run to list Streams administrators and other users who allow access to remote Streams administrators:

- Listing Local Streams Administrators
- Listing Users Who Allow Access to Remote Streams Administrators

See Also: *Oracle Database PL/SQL Packages and Types Reference* for more information about configuring Streams administrators and other Streams users using the `DBMS_STREAMS_AUTH` package

Listing Local Streams Administrators

You optionally can grant privileges to a local Streams administrator by running the GRANT_ADMIN_PRIVILEGE procedure in the DBMS_STREAMS_AUTH package. The DBA_STREAMS_ADMINISTRATOR data dictionary view contains only the local Streams administrators created with the grant_privileges parameter set to true when the GRANT_ADMIN_PRIVILEGE procedure was run for the user. If you created a Streams administrator using generated scripts and set the grant_privileges parameter to false when the GRANT_ADMIN_PRIVILEGE procedure was run for the user, then the DBA_STREAMS_ADMINISTRATOR data dictionary view does not list the user as a Streams administrator.

To list the local Streams administrators created with the grant_privileges parameter set to true when running the GRANT_ADMIN_PRIVILEGE procedure, run the following query:

```
COLUMN USERNAME HEADING 'Local Streams Administrator' FORMAT A30

SELECT USERNAME FROM DBA_STREAMS_ADMINISTRATOR
  WHERE LOCAL_PRIVILEGES = 'YES';
```

Your output looks similar to the following:

```
Local Streams Administrator
------------------------------
STRMADMIN
```

The GRANT_ADMIN_PRIVILEGE might not have been run on a user who is a Streams administrator. Such administrators are not returned by the query in this section. Also, you can change the privileges for the users listed after the GRANT_ADMIN_PRIVILEGE procedure has been run for them. The DBA_STREAMS_ADMINISTRATOR view does not track these changes unless they are performed by the DBMS_STREAMS_AUTH package. For example, you can revoke the privileges granted by the GRANT_ADMIN_PRIVILEGE procedure for a particular user using the REVOKE SQL statement, but this user would be listed when you query the DBA_STREAMS_ADMINISTRATOR view.

Oracle recommends using the REVOKE_ADMIN_PRIVILEGE procedure in the DBMS_STREAMS_AUTH package to revoke privileges from a user listed by the query in this section. When you revoke privileges from a user using this procedure, the user is removed from the DBA_STREAMS_ADMINISTRATOR view.

See Also: "Configuring a Streams Administrator" on page 10-1

Listing Users Who Allow Access to Remote Streams Administrators

You can configure a user to allow access to remote Streams administrators by running the GRANT_REMOTE_ADMIN_ACCESS procedure in the DBMS_STREAMS_AUTH package. Such a user allows the remote Streams administrator to perform administrative actions in the local database using a database link.

Typically, you configure such a user at a local **source database** if a **downstream capture process** captures changes originating at the local source database. The Streams administrator at a downstream capture database administers the source database using this connection.

To list the users who allow to remote Streams administrators, run the following query:

```
COLUMN USERNAME HEADING 'Users Who Allow Remote Access' FORMAT A30

SELECT USERNAME FROM DBA_STREAMS_ADMINISTRATOR
  WHERE ACCESS_FROM_REMOTE = 'YES';
```

Your output looks similar to the following:

```
Users Who Allow Remote Access
-----------------------------
STRMREMOTE
```

Monitoring the Streams Pool

The **Streams pool** is a portion of memory in the SGA that is used by Streams. The Streams pool stores enqueued **message**s in memory, and it provides memory for **capture process**es and **apply process**es. The Streams pool always stores LCRs captured by a capture process, and it can store **user-enqueued message**s.

The Streams pool size is managed by Automatic Shared Memory Management when the SGA_TARGET initialization parameter is set to a nonzero value. If this parameter is set to 0 (zero), then you can specify the size of the Streams pool in bytes using the STREAMS_POOL_SIZE initialization parameter. In this case, the V$STREAMS_POOL_ADVICE dynamic performance view provides information about an appropriate setting for the STREAMS_POOL_SIZE initialization parameter.

This section contains example queries that show when you should increase, retain, or decrease the size of the Streams pool. Each query shows the following information about the Streams pool:

- STREAMS_POOL_SIZE_FOR_ESTIMATE shows the size, in megabytes, of the Streams pool for the estimate. The size ranges from values smaller than the current Streams pool size to values larger than the current Streams pool size, and there is a separate row for each increment. There always is an entry that shows the current Streams pool size, and there always are 20 increments. The range and the size of the increments depend on the current size of the Streams pool.
- STREAMS_POOL_SIZE_FACTOR shows the size factor of an estimate as it relates to the current size of the Streams pool. For example, a size factor of .2 means that the estimate is for 20% of the current size of the Streams pool, while a size factor of 1.6 means that the estimate is for 160% of the current size of the Streams pool. The row with a size factor of 1.0 shows the current size of the Streams pool.
- ESTD_SPILL_COUNT shows the estimated number messages that will spill from memory to the **queue table** for each STREAMS_POOL_SIZE_FOR_ESTIMATE and STREAMS_POOL_SIZE_FACTOR returned by the query.
- ESTD_SPILL_TIME shows the estimated elapsed time, in seconds, spent spilling messages from memory to the queue table for each STREAMS_POOL_SIZE_FOR_ESTIMATE and STREAMS_POOL_SIZE_FACTOR returned by the query.
- ESTD_UNSPILL_COUNT shows the estimated number messages that will unspill from the queue table back into memory for each STREAMS_POOL_SIZE_FOR_ESTIMATE and STREAMS_POOL_SIZE_FACTOR returned by the query.
- ESTD_UNSPILL_TIME shows the estimated elapsed time, in seconds, spent unspilling messages from the queue table back into memory for each STREAMS_POOL_SIZE_FOR_ESTIMATE and STREAMS_POOL_SIZE_FACTOR returned by the query.

If any capture processes, **propagation**s, or apply processes are disabled when you query the V$STREAMS_POOL_ADVICE view, and you plan to enable them in the future, then make sure you consider the memory resources required by these **Streams client**s before you decrease the size of the Streams pool.

Tips:

- In general, the best size for the Streams pool is the smallest size for which spilled and unspilled messages and times are close to zero.
- For the most accurate results, you should run a query on the V$STREAMS_POOL_ADVICE view when there is a normal amount of dequeue activity by propagations and apply processes in a database. If dequeue activity is far lower than normal, or far higher than normal, then the query results might not be a good guide for adjusting the size of the Streams pool.

See Also:

- Streams Pool on page 3-19
- "Setting Initialization Parameters Relevant to Streams" on page 10-4 for more information about the STREAMS_POOL_SIZE initialization parameter

Query Result that Advises Increasing the Streams Pool Size

Consider the following results returned by the V$STREAMS_POOL_ADVICE view:

```
COLUMN STREAMS_POOL_SIZE_FOR_ESTIMATE HEADING 'Streams Pool Size|for Estimate(MB)'
  FORMAT 999999999999
COLUMN STREAMS_POOL_SIZE_FACTOR HEADING 'Streams Pool|Size|Factor' FORMAT 99.9
COLUMN ESTD_SPILL_COUNT HEADING 'Estimated|Spill|Count' FORMAT 99999999
COLUMN ESTD_SPILL_TIME HEADING 'Estimated|Spill|Time' FORMAT 99999999.99
COLUMN ESTD_UNSPILL_COUNT HEADING 'Estimated|Unspill|Count' FORMAT 99999999
COLUMN ESTD_UNSPILL_TIME HEADING 'Estimated|Unspill|Time' FORMAT 99999999.99

SELECT STREAMS_POOL_SIZE_FOR_ESTIMATE,
       STREAMS_POOL_SIZE_FACTOR,
       ESTD_SPILL_COUNT,
       ESTD_SPILL_TIME,
       ESTD_UNSPILL_COUNT,
       ESTD_UNSPILL_TIME
  FROM V$STREAMS_POOL_ADVICE;
```

Streams Pool Size for Estimate(MB)	Streams Pool Size Factor	Estimated Spill Count	Estimated Spill Time	Estimated Unspill Count	Estimated Unspill Time
24	.1	158	62.00	0	.00
48	.2	145	59.00	0	.00
72	.3	137	53.00	0	.00
96	.4	122	50.00	0	.00
120	.5	114	48.00	0	.00
144	.6	103	45.00	0	.00
168	.7	95	39.00	0	.00
192	.8	87	32.00	0	.00
216	.9	74	26.00	0	.00
240	1.0	61	21.00	0	.00
264	1.1	56	17.00	0	.00
288	1.2	43	15.00	0	.00
312	1.3	36	11.00	0	.00
336	1.4	22	8.00	0	.00

```
               360          1.5          9         2.00          0          .00
               384          1.6          0          .00          0          .00
               408          1.7          0          .00          0          .00
               432          1.8          0          .00          0          .00
               456          1.9          0          .00          0          .00
               480          2.0          0          .00          0          .00
```

Based on these results, 384 megabytes, or 160% of the size of the current **Streams pool**, is the optimal size for the Streams pool. That is, this size is the smallest size for which the estimated number of spilled and unspilled **message**s is zero.

> **Note:** After you adjust the size of the Streams pool, it might take some time for the new size to result in new output for the `V$STREAMS_POOL_ADVICE` view.

Query Result that Advises Retaining the Current Streams Pool Size

Consider the following results returned by the `V$STREAMS_POOL_ADVICE` view:

```
COLUMN STREAMS_POOL_SIZE_FOR_ESTIMATE  HEADING 'Streams Pool|Size for Estimate'
  FORMAT 999999999999
COLUMN STREAMS_POOL_SIZE_FACTOR HEADING 'Streams Pool|Size|Factor' FORMAT 99.9
COLUMN ESTD_SPILL_COUNT HEADING 'Estimated|Spill|Count' FORMAT 99999999
COLUMN ESTD_SPILL_TIME HEADING 'Estimated|Spill|Time' FORMAT 99999999.99
COLUMN ESTD_UNSPILL_COUNT HEADING 'Estimated|Unspill|Count' FORMAT 99999999
COLUMN ESTD_UNSPILL_TIME HEADING 'Estimated|Unspill|Time' FORMAT 99999999.99

SELECT STREAMS_POOL_SIZE_FOR_ESTIMATE,
       STREAMS_POOL_SIZE_FACTOR,
       ESTD_SPILL_COUNT,
       ESTD_SPILL_TIME,
       ESTD_UNSPILL_COUNT,
       ESTD_UNSPILL_TIME
  FROM V$STREAMS_POOL_ADVICE;

                  Streams Pool Estimated    Estimated Estimated    Estimated
Streams Pool Size         Size     Spill        Spill   Unspill      Unspill
 for Estimate(MB)       Factor     Count         Time     Count         Time
----------------- ------------ --------- ------------ --------- ------------
               24           .1        89        52.00         0          .00
               48           .2        78        48.00         0          .00
               72           .3        71        43.00         0          .00
               96           .4        66        37.00         0          .00
              120           .5        59        32.00         0          .00
              144           .6        52        26.00         0          .00
              168           .7        39        20.00         0          .00
              192           .8        27        12.00         0          .00
              216           .9        15         5.00         0          .00
              240          1.0         0          .00         0          .00
              264          1.1         0          .00         0          .00
              288          1.2         0          .00         0          .00
              312          1.3         0          .00         0          .00
              336          1.4         0          .00         0          .00
              360          1.5         0          .00         0          .00
              384          1.6         0          .00         0          .00
              408          1.7         0          .00         0          .00
              432          1.8         0          .00         0          .00
              456          1.9         0          .00         0          .00
              480          2.0         0          .00         0          .00
```

Based on these results, the current size of the **Streams pool** is the optimal size. That is, this size is the smallest size for which the estimated number of spilled and unspilled **message**s is zero.

Query Result that Advises Decreasing the Streams Pool Size

Consider the following results returned by the V$STREAMS_POOL_ADVICE view:

```
COLUMN STREAMS_POOL_SIZE_FOR_ESTIMATE  HEADING 'Streams Pool|Size for Estimate'
  FORMAT 999999999999
COLUMN STREAMS_POOL_SIZE_FACTOR HEADING 'Streams Pool|Size|Factor' FORMAT 99.9
COLUMN ESTD_SPILL_COUNT HEADING 'Estimated|Spill|Count' FORMAT 99999999
COLUMN ESTD_SPILL_TIME HEADING 'Estimated|Spill|Time' FORMAT 99999999.99
COLUMN ESTD_UNSPILL_COUNT HEADING 'Estimated|Unspill|Count' FORMAT 99999999
COLUMN ESTD_UNSPILL_TIME HEADING 'Estimated|Unspill|Time' FORMAT 99999999.99

SELECT STREAMS_POOL_SIZE_FOR_ESTIMATE,
       STREAMS_POOL_SIZE_FACTOR,
       ESTD_SPILL_COUNT,
       ESTD_SPILL_TIME,
       ESTD_UNSPILL_COUNT,
       ESTD_UNSPILL_TIME
  FROM V$STREAMS_POOL_ADVICE;
```

Streams Pool Size for Estimate(MB)	Streams Pool Size Factor	Estimated Spill Count	Estimated Spill Time	Estimated Unspill Count	Estimated Unspill Time
24	.1	158	62.00	0	.00
48	.2	145	59.00	0	.00
72	.3	137	53.00	0	.00
96	.4	122	50.00	0	.00
120	.5	114	48.00	0	.00
144	.6	103	45.00	0	.00
168	.7	0	.00	0	.00
192	.8	0	.00	0	.00
216	.9	0	.00	0	.00
240	1.0	0	.00	0	.00
264	1.1	0	.00	0	.00
288	1.2	0	.00	0	.00
312	1.3	0	.00	0	.00
336	1.4	0	.00	0	.00
360	1.5	0	.00	0	.00
384	1.6	0	.00	0	.00
408	1.7	0	.00	0	.00
432	1.8	0	.00	0	.00
456	1.9	0	.00	0	.00
480	2.0	0	.00	0	.00

Based on these results, 168 megabytes, or 70% of the size of the current **Streams pool**, is the optimal size for the Streams pool. That is, this size is the smallest size for which the estimated number of spilled and unspilled messages is zero.

Note: After you adjust the size of the Streams pool, it might take some time for the new size to result in new output for the V$STREAMS_POOL_ADVICE view.

Monitoring Compatibility in a Streams Environment

The queries in the following sections show Streams compatibility for tables in the local database:

- Listing the Database Objects that Are Not Compatible with Streams
- Listing the Database Objects that Have Become Compatible with Streams Recently

Listing the Database Objects that Are Not Compatible with Streams

A database object is not compatible with Streams if a **capture process** cannot capture changes to the object. The query in this section displays the following information about objects that are not compatible with Streams:

- The object owner
- The object name
- The reason why the object is not compatible with Streams
- Whether capture processes automatically filter out changes to the object (`AUTO_FILTERED` column)

If capture processes automatically filter out changes to an object, then the rules sets used by the capture processes do not need to filter them out explicitly. For example, capture processes automatically filter out changes to materialized view logs. However, if changes to incompatible objects are not filtered out automatically, then the **rule set**s used by each capture process must filter them out to avoid errors.

For example, suppose the rule sets for a capture process instruct the capture process to capture all of the changes made to a specific schema. The query in this section shows that one object in this schema is not compatible with Streams, and that changes to the object are not filtered out automatically. In this case, you can add a **rule** to the **negative rule set** for the capture process to filter out changes to the incompatible object.

The `AUTO_FILTERED` column pertains only to capture processes. Apply processes do not automatically filter out LCRs that encapsulate changes to objects that are not compatible with Streams, even if the `AUTO_FILTERED` column is `YES` for the object. Such changes can result in apply errors if they are dequeued by an **apply process**.

Run the following query to list the objects in the local database that are not compatible with Streams:

```
COLUMN OWNER HEADING 'Object|Owner' FORMAT A8
COLUMN TABLE_NAME HEADING 'Object Name' FORMAT A30
COLUMN REASON HEADING 'Reason' FORMAT A30
COLUMN AUTO_FILTERED HEADING 'Auto|Filtered?' FORMAT A9

SELECT OWNER, TABLE_NAME, REASON, AUTO_FILTERED FROM DBA_STREAMS_UNSUPPORTED;
```

Your output looks similar to the following:

```
Object                                                                  Auto
Owner    Object Name                    Reason                          Filtered?
-------- ------------------------------ ------------------------------- ---------
HR       MLOG$_COUNTRIES                materialized view log           YES
HR       MLOG$_DEPARTMENTS              materialized view log           YES
HR       MLOG$_EMPLOYEES                materialized view log           YES
HR       MLOG$_JOBS                     materialized view log           YES
HR       MLOG$_JOB_HISTORY              materialized view log           YES
HR       MLOG$_LOCATIONS                materialized view log           YES
HR       MLOG$_REGIONS                  materialized view log           YES
IX       AQ$_ORDERS_QUEUETABLE_G        IOT with overflow               NO
IX       AQ$_ORDERS_QUEUETABLE_H        unsupported column exists       NO
IX       AQ$_ORDERS_QUEUETABLE_I        unsupported column exists       NO
IX       AQ$_ORDERS_QUEUETABLE_S        AQ queue table                  NO
IX       AQ$_ORDERS_QUEUETABLE_T        AQ queue table                  NO
IX       ORDERS_QUEUETABLE              column with user-defined type   NO
OE       CATEGORIES_TAB                 column with user-defined type   NO
OE       CUSTOMERS                      column with user-defined type   NO
OE       PRODUCT_REF_LIST_NESTEDTAB     column with user-defined type   NO
OE       SUBCATEGORY_REF_LIST_NESTEDTAB column with user-defined type   NO
OE       WAREHOUSES                     column with user-defined type   NO
PM       ONLINE_MEDIA                   column with user-defined type   NO
PM       PRINT_MEDIA                    column with user-defined type   NO
PM       TEXTDOCS_NESTEDTAB             column with user-defined type   NO
SH       MVIEW$_EXCEPTIONS              unsupported column exists       NO
SH       SALES_TRANSACTIONS_EXT         external table                  NO
```

Notice that the `Auto Filtered?` column is `YES` for the `oe.mlog$_orders` materialized view log. Each capture process automatically filters out changes to this object, even if the rules sets for a capture process instruct the capture process to capture changes to the object.

Because the `Auto Filtered?` column is `NO` for the other objects listed in the example output, capture processes do not filter out changes to these objects automatically. If a capture process attempts to process LCRs for these unsupported objects, then the capture process raises an error. However, you can avoid these errors by configuring rules sets that instruct the capture process not to capture changes to these unsupported objects.

Note: The results of the query in this section depend on the compatibility level of the database. More database objects are incompatible with Streams at lower compatibility levels. The `COMPATIBLE` initialization parameter controls the compatibility level of the database.

See Also:

- Chapter 6, "How Rules Are Used in Streams"
- *Oracle Database Reference* and *Oracle Database Upgrade Guide* for more information about the `COMPATIBLE` initialization parameter

Listing the Database Objects that Have Become Compatible with Streams Recently

The query in this section displays the following information about database objects that have become compatible with Streams in a recent release of Oracle:

- The object owner
- The object name
- The reason why the object was not compatible with Streams in previous releases of Oracle
- The Oracle release in which the object became compatible with Streams

Run the following query to display this information for the local database:

```
COLUMN OWNER HEADING 'Owner' FORMAT A10
COLUMN TABLE_NAME HEADING 'Object Name' FORMAT A20
COLUMN REASON HEADING 'Reason' FORMAT A30
COLUMN COMPATIBLE HEADING 'Compatible' FORMAT A10

SELECT OWNER, TABLE_NAME, REASON, COMPATIBLE FROM DBA_STREAMS_NEWLY_SUPPORTED;
```

Your output looks similar to the following:

```
Owner      Object Name          Reason                         Compatible
---------- -------------------- ------------------------------ ----------
HR         COUNTRIES            IOT                            10.1
OUTLN      OL$                  unsupported column exists      10.1
SH         CAL_MONTH_SALES_MV   unsupported column exists      10.1
SH         FWEEK_PSCAT_SALES_MV unsupported column exists      10.1
SH         PLAN_TABLE           unsupported column exists      10.1
DBSNMP     MGMT_BSLN_RAWDATA    IOT                            10.1
HR         COUNTRIES            IOT                            10.1
IX         AQ$_ORDERS_QUEUETABL IOT with overflow              10.2
           E_G
OUTLN      OL$                  unsupported column exists      10.1
SH         CAL_MONTH_SALES_MV   unsupported column exists      10.1
SH         FWEEK_PSCAT_SALES_MV unsupported column exists      10.1
SH         PLAN_TABLE           unsupported column exists      10.1
STRMADMIN  AQ$_STREAMS_QUEUE_TA IOT with overflow              10.2
           BLE_D
STRMADMIN  AQ$_STREAMS_QUEUE_TA IOT with overflow              10.2
           BLE_G
WMSYS      AQ$_WM$EVENT_QUEUE_T IOT with overflow              10.2
           ABLE_G
```

The `Compatible` column shows the minimum database compatibility for Streams to support the object. If the local database compatibility is equal to or higher than the value in the `Compatible` column for an object, then **capture process**es and **apply process**es can process changes to the object successfully. You control the compatibility of an Oracle database using the `COMPATIBLE` initialization parameter.

If your Streams environment includes databases that are running different versions of the Oracle Database, then you can configure **rule**s that use the `GET_COMPATIBLE` member function for LCRs to filter out LCRs that are not compatible with particular databases. These rules can be added to the **rule set**s of capture processes, **propagation**s, and apply processes to filter out incompatible LCRs wherever necessary in a stream.

See Also:

- *Oracle Database Reference* and *Oracle Database Upgrade Guide* for more information about the `COMPATIBLE` initialization parameter
- "Rule Conditions that Instruct Streams Clients to Discard Unsupported LCRs" on page 6-42 for information about creating rules that use the `GET_COMPATIBLE` member function for LCRs
- "Listing the Database Objects that Are Not Compatible with Streams" on page 26-7 for more information about objects that are not compatible with Streams

Monitoring Streams Performance Using AWR and Statspack

You can use Automatic Workload Repository (AWR) to monitor performance statistics related to Streams. If AWR is not available on your database, then you can use the Statspack package to monitor performance statistics related to Streams. The most current instructions and information on installing and using the Statspack package are contained in the `spdoc.txt` file installed with your database. Refer to that file for Statspack information. On Unix systems, the file is located in the *`ORACLE_HOME`*`/rdbms/admin` directory. On Windows systems, the file is located in the *`ORACLE_HOME`*`\rdbms\admin` directory.

See Also: *Oracle Database Performance Tuning Guide* for more information about AWR

Part IV

Sample Environments and Applications

This part includes the following detailed examples:

- Chapter 27, "Single-Database Capture and Apply Example"
- Chapter 28, "Rule-Based Application Example"

27

Single-Database Capture and Apply Example

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

This chapter contains these topics:

Note: The extended example is not included in the PDF version of this chapter, but it is included in the HTML version of the chapter.

Overview of the Single-Database Capture and Apply Example

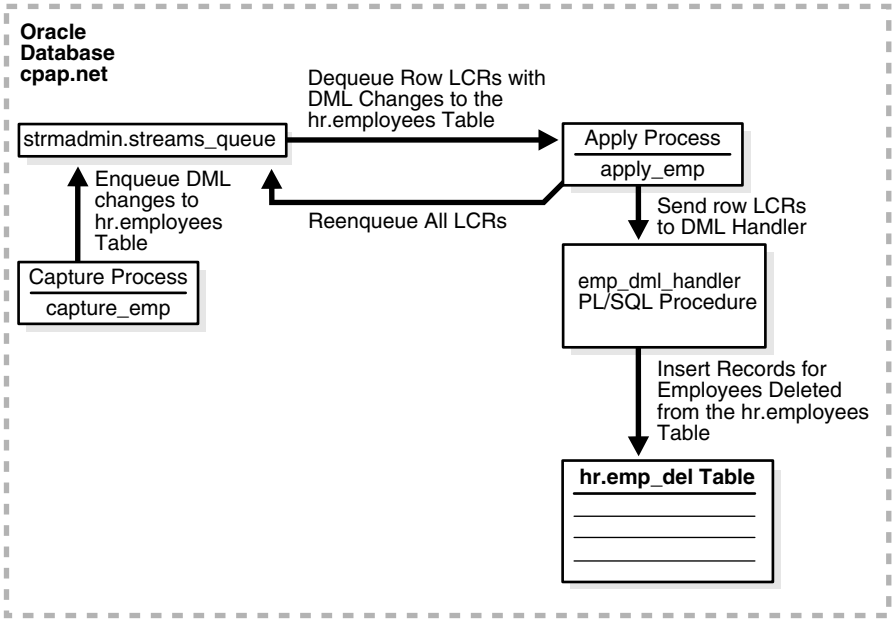
The example in this chapter illustrates using Streams to capture and apply data manipulation language (DML) changes at a single database named `cpap.net`. Specifically, this example captures DML changes to the `employees` table in the `hr` schema, placing row logical change records (LCRs) into a **queue**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

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

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

Figure 27–1 provides an overview of the environment.

Figure 27–1 Single Database Capture and Apply Example



See Also:

- Chapter 2, "Streams Capture Process"
- "LCR Processing" on page 4-4 for more information about DML handlers

Prerequisites

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

- Set the following initialization parameters to the values indicated for all databases in the environment:
  - Set the `COMPATIBLE` initialization parameter to `10.1.0` or higher.
  - `STREAMS_POOL_SIZE`: Optionally set this parameter to an appropriate value for each database in the environment. This parameter specifies the size of the **Streams pool**. The Streams pool stores **message**s in a **buffered queue** and is used for internal communications during parallel capture and apply. When the `SGA_TARGET` initialization parameter is set to a nonzero value, the Streams pool size is managed by Automatic Shared Memory Management.

    See Also: "Setting Initialization Parameters Relevant to Streams" on page 10-4 for information about other initialization parameters that are important in a Streams environment

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

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

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

28

Rule-Based Application Example

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

The examples in this chapter are independent of Streams. That is, no Streams **capture process**es, **propagation**s, **apply process**es, or **messaging client**s are clients of the rules engine in these examples, and no **queue**s are used.

This chapter contains these topics:

Note: The extended example is not included in the PDF version of this chapter, but it is included in the HTML version of the chapter.

See Also:

- Chapter 5, "Rules"
- Chapter 14, "Managing Rules"
- Chapter 23, "Monitoring Rules"

Overview of the Rule-Based Application

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

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

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

Part V

Appendixes

This part includes the following appendix:

- Appendix A, "XML Schema for LCRs"
- Appendix B, "Online Database Upgrade with Streams"
- Appendix C, "Online Database Maintenance with Streams"

A

XML Schema for LCRs

The XML schema described in this appendix defines the format of a **logical change record (LCR)**. The Oracle XML DB must be installed to use the XML schema for LCRs.

This appendix contains this topic:

- Definition of the XML Schema for LCRs

The namespace for this schema is the following:

```
http://xmlns.oracle.com/streams/schemas/lcr
```

The schema is the following:

```
http://xmlns.oracle.com/streams/schemas/lcr/streamslcr.xsd
```

> **See Also:** *Oracle XML DB Developer's Guide* for more information about Oracle XML DB and for information about upgrading an existing XML schema for LCRs

Definition of the XML Schema for LCRs

The following is the XML schema definition for LCRs:

```
'<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://xmlns.oracle.com/streams/schemas/lcr"
        xmlns:lcr="http://xmlns.oracle.com/streams/schemas/lcr"
        xmlns:xdb="http://xmlns.oracle.com/xdb"
          version="1.0"
        elementFormDefault="qualified">

  <simpleType name = "short_name">
    <restriction base = "string">
      <maxLength value="30"/>
    </restriction>
  </simpleType>

  <simpleType name = "long_name">
    <restriction base = "string">
      <maxLength value="4000"/>
    </restriction>
  </simpleType>

  <simpleType name = "db_name">
    <restriction base = "string">
      <maxLength value="128"/>
    </restriction>
```

```
</simpleType>

<!-- Default session parameter is used if format is not specified -->
<complexType name="datetime_format">
  <sequence>
    <element name = "value" type = "string" nillable="true"/>
    <element name = "format" type = "string" minOccurs="0" nillable="true"/>
  </sequence>
</complexType>

<complexType name="anydata">
  <choice>
    <element name="varchar2" type = "string" xdb:SQLType="CLOB"
                                                    nillable="true"/>

    <!-- Represent char as varchar2. xdb:CHAR blank pads upto 2000 bytes! -->
    <element name="char" type = "string" xdb:SQLType="CLOB"
                                                    nillable="true"/>
    <element name="nchar" type = "string" xdb:SQLType="NCLOB"
                                                    nillable="true"/>

    <element name="nvarchar2" type = "string" xdb:SQLType="NCLOB"
                                                    nillable="true"/>
    <element name="number" type = "double" xdb:SQLType="NUMBER"
                                                    nillable="true"/>
    <element name="raw" type = "hexBinary" xdb:SQLType="BLOB"
                                                    nillable="true"/>
    <element name="date" type = "lcr:datetime_format"/>
    <element name="timestamp" type = "lcr:datetime_format"/>
    <element name="timestamp_tz" type = "lcr:datetime_format"/>
    <element name="timestamp_ltz" type = "lcr:datetime_format"/>

    <!-- Interval YM should be as per format allowed by SQL -->
    <element name="interval_ym" type = "string" nillable="true"/>

    <!-- Interval DS should be as per format allowed by SQL -->
    <element name="interval_ds" type = "string" nillable="true"/>

    <element name="urowid" type = "string" xdb:SQLType="VARCHAR2"
                                                    nillable="true"/>
  </choice>
</complexType>

<complexType name="column_value">
  <sequence>
    <element name = "column_name" type = "lcr:long_name" nillable="false"/>
    <element name = "data" type = "lcr:anydata" nillable="false"/>
    <element name = "lob_information" type = "string" minOccurs="0"
                                                      nillable="true"/>
    <element name = "lob_offset" type = "nonNegativeInteger" minOccurs="0"
                                                      nillable="true"/>
    <element name = "lob_operation_size" type = "nonNegativeInteger"
                                          minOccurs="0" nillable="true"/>
    <element name = "long_information" type = "string" minOccurs="0"
                                                      nillable="true"/>
  </sequence>
</complexType>
```

```
<complexType name="extra_attribute">
  <sequence>
    <element name = "attribute_name" type = "lcr:short_name"/>
    <element name = "attribute_value" type = "lcr:anydata"/>
  </sequence>
</complexType>

<element name = "ROW_LCR" xdb:defaultTable="">
  <complexType>
    <sequence>
      <element name = "source_database_name" type = "lcr:db_name"
                                                    nillable="false"/>
      <element name = "command_type" type = "string" nillable="false"/>
      <element name = "object_owner" type = "lcr:short_name"
                                                    nillable="false"/>
      <element name = "object_name" type = "lcr:short_name"
                                                    nillable="false"/>
      <element name = "tag" type = "hexBinary" xdb:SQLType="RAW"
                                              minOccurs="0" nillable="true"/>
      <element name = "transaction_id" type = "string" minOccurs="0"
                                               nillable="true"/>
      <element name = "scn" type = "double" xdb:SQLType="NUMBER"
                                              minOccurs="0" nillable="true"/>
      <element name = "old_values" minOccurs = "0">
        <complexType>
          <sequence>
            <element name = "old_value" type="lcr:column_value"
                                                 maxOccurs = "unbounded"/>
          </sequence>
        </complexType>
      </element>
      <element name = "new_values" minOccurs = "0">
        <complexType>
          <sequence>
            <element name = "new_value" type="lcr:column_value"
                                                 maxOccurs = "unbounded"/>
          </sequence>
        </complexType>
      </element>
      <element name = "extra_attribute_values" minOccurs = "0">
        <complexType>
          <sequence>
            <element name = "extra_attribute_value"
                     type="lcr:extra_attribute"
                     maxOccurs = "unbounded"/>
          </sequence>
        </complexType>
      </element>
    </sequence>
  </complexType>
</element>

<element name = "DDL_LCR" xdb:defaultTable="">
  <complexType>
    <sequence>
      <element name = "source_database_name" type = "lcr:db_name"
                                                   nillable="false"/>
      <element name = "command_type" type = "string" nillable="false"/>
      <element name = "current_schema" type = "lcr:short_name"
                                                   nillable="false"/>
```

```
        <element name = "ddl_text" type = "string" xdb:SQLType="CLOB"
                                                    nillable="false"/>
        <element name = "object_type" type = "string"
                                    minOccurs = "0" nillable="true"/>
        <element name = "object_owner" type = "lcr:short_name"
                                    minOccurs = "0" nillable="true"/>
        <element name = "object_name" type = "lcr:short_name"
                                    minOccurs = "0" nillable="true"/>
        <element name = "logon_user" type = "lcr:short_name"
                                    minOccurs = "0" nillable="true"/>
        <element name = "base_table_owner" type = "lcr:short_name"
                                    minOccurs = "0" nillable="true"/>
        <element name = "base_table_name" type = "lcr:short_name"
                                    minOccurs = "0" nillable="true"/>
        <element name = "tag" type = "hexBinary" xdb:SQLType="RAW"
                                    minOccurs = "0" nillable="true"/>
        <element name = "transaction_id" type = "string"
                                    minOccurs = "0" nillable="true"/>
        <element name = "scn" type = "double" xdb:SQLType="NUMBER"
                                    minOccurs = "0" nillable="true"/>
        <element name = "extra_attribute_values" minOccurs = "0">
          <complexType>
            <sequence>
              <element name = "extra_attribute_value"
                       type="lcr:extra_attribute"
                       maxOccurs = "unbounded"/>
            </sequence>
          </complexType>
        </element>
      </sequence>
    </complexType>
  </element>
</schema>';
```

B

Online Database Upgrade with Streams

This appendix describes how to perform a database upgrade of an Oracle Database with Oracle Streams. The database upgrade operation described in this appendix uses the features of Oracle Streams to achieve little or no database down time. To use Streams for a database upgrade, the database must be Oracle9*i* Database Release 2 (9.2) or later.

This appendix contains these topics:

- Overview of Using Streams in the Database Upgrade Process
- Preparing for a Database Upgrade Using Streams
- Performing a Database Upgrade Using Streams

> **See Also:** Appendix C, "Online Database Maintenance with Streams" for information about performing other database maintenance operations with Streams

Overview of Using Streams in the Database Upgrade Process

An Oracle database upgrade is the process of transforming an existing, prior release of an Oracle Database system into the current release of the Oracle Database system. A database upgrade typically requires substantial database down time, but you can perform a database upgrade with little or no down time by using the features of Oracle Streams. To do so, you use Oracle Streams to configure a single-source **replication** environment with the following databases:

- **Source Database:** The original database that is being upgraded.
- **Capture Database:** The database where a **capture process** captures changes made to the source database during the upgrade.
- **Destination Database:** The copy of the source database where an **apply process** applies changes made to the source database during the upgrade process.

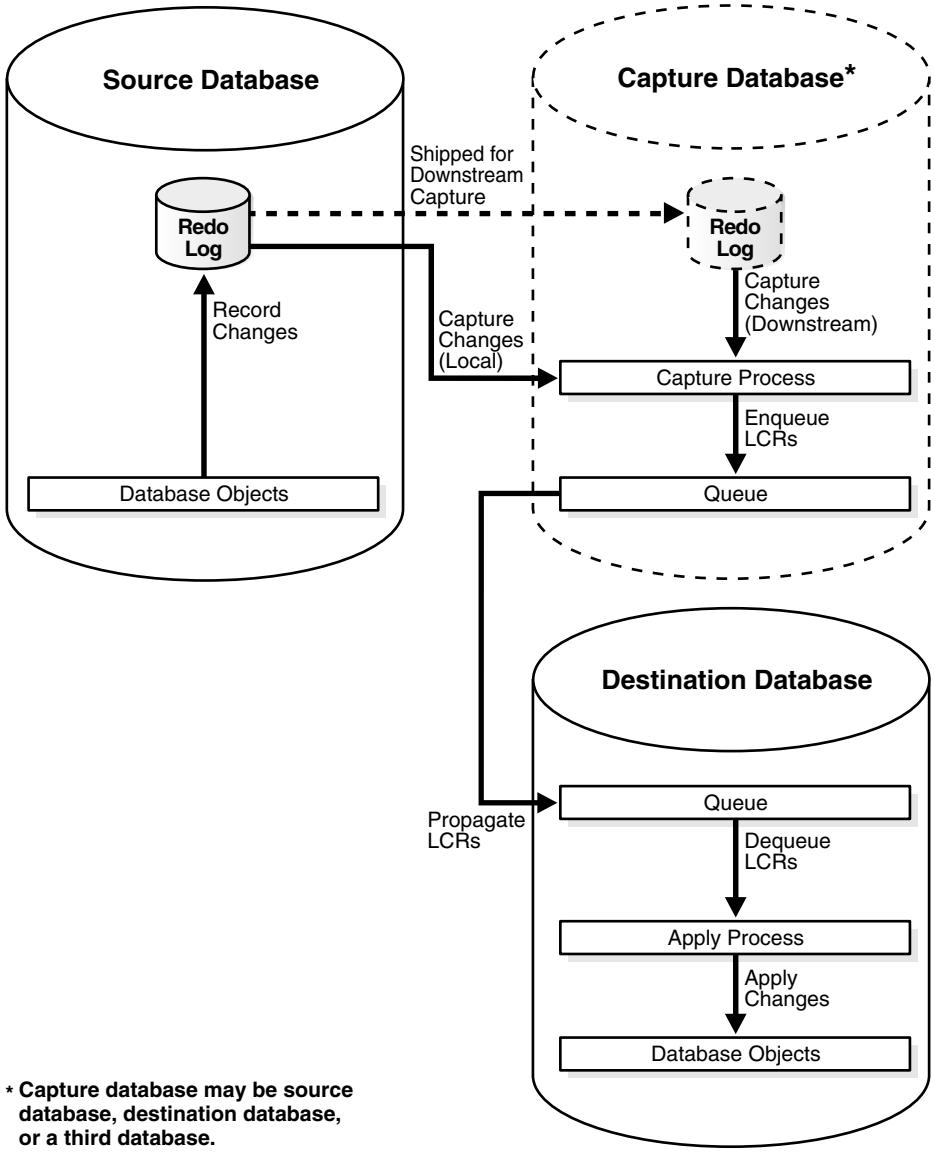
Specifically, you can use the following general steps to perform a database upgrade while the database is online:

1. Create an empty **destination database**.
2. Configure an Oracle Streams single-source replication environment where the original database is the source database and a copy of the database is the destination database for the changes made at the source.
3. Perform the database upgrade on the destination database. During this time the original source database is available online.

4. Use Oracle Streams to apply the changes made at the source database to the destination database.
5. When the destination database has caught up with the changes made at the source database, take the source database offline and make the destination database available for applications and users.

Figure B–1 provides an overview of this process.

Figure B–1 Online Database Upgrade with Streams



The Capture Database During the Upgrade Process

During the upgrade process, the capture database is the database where the **capture process** is created. Downstream capture was introduced in Oracle Database 10*g* Release 1 (10.1). If you are upgrading a database from Oracle Database 10*g* Release 1 to Oracle Database 10*g* Release 2 (10.2), then you have the following options:

- A **local capture process** can be created at the source database during the upgrade process.
- A **downstream capture process** can be created at the **destination database**. If the destination database is the capture database, then a **propagation** from the capture database to the destination database is not needed.
- A third database can be the capture database. In this case, the third database can be Oracle Database 10*g* Release 1 or Oracle Database 10*g* Release 2.

However, if you are upgrading a database from Oracle9*i* Database Release 2 (9.2) to Oracle Database 10*g* Release 2, then downstream capture is not supported, and a local capture process must be created at the source database.

A downstream capture process reduces the resources required at the source database during the upgrade process, but a local capture process is easier to configure. Table B–1 describes which database can be the capture database during the upgrade process.

Table B–1 Supported Capture Database During Upgrade

Existing Database Release	Capture Database Can Be Source Database?	Capture Database Can Be Destination Database?	Capture Database Can Be Third Database?
9.2	Yes	No	No
10.1	Yes	Yes	Yes

Note: If you are upgrading from Oracle Database 10*g* Release 1, then, before you begin the upgrade, decide which database will be the capture database.

See Also: "Local Capture and Downstream Capture" on page 2-12

Assumptions for the Database Being Upgraded

The instructions in this appendix assume that all of the following statements are true for the database being upgraded:

- The database is not part of an existing Oracle Streams environment.
- The database is not part of an existing logical standby environment.
- The database is not part of an existing Advanced Replication environment.
- No tables at the database are master tables for materialized views in other databases.
- Any user-created **queue**s are read-only during the upgrade process.

Considerations for Job Queue Processes and PL/SQL Package Subprograms

If possible, ensure that no job queue processes are created, modified, or deleted during the upgrade process, and that no Oracle-supplied PL/SQL package subprograms are invoked during the upgrade process that modify both user data and dictionary metadata at the same time. The following packages contain subprograms that modify both user data and dictionary metadata at the same time: `DBMS_RLS`, `DBMS_STATS`, and `DBMS_JOB`.

It might be possible to perform such actions on the database if you ensure that the same actions are performed on the source database and **destination database** in Steps 9 and 10 in "Task 5: Finishing the Upgrade and Removing Streams" on page B-19. For example, if a PL/SQL procedure gathers statistics on the source database during the upgrade process, then the same PL/SQL procedure should be invoked at the destination database in Step 10.

Preparing for a Database Upgrade Using Streams

The following sections describe tasks to complete before starting the database upgrade with Streams:

- Preparing to Upgrade a Database with User-defined Types
- Deciding Which Utility to Use for Instantiation

Preparing to Upgrade a Database with User-defined Types

User-defined types include object types, `REF` values, varrays, and nested tables. Currently, Streams **capture process**es and **apply process**es do not support user-defined types. This section discusses using Streams to perform a database upgrade on a database that has user-defined types.

One option is to make tables that contain user-defined types read-only during the database upgrade. In this case, these tables are instantiated at the **destination database**, and no changes are made to these tables during the entire operation. After the upgrade is complete, make the tables that contain user-defined types read/write at the destination database.

If tables that contain user-defined types must remain open during the upgrade, then the following general steps can be used to retain changes to these tables during the upgrade:

1. Before you begin the upgrade process described in "Performing a Database Upgrade Using Streams" on page B-6, create one or more logging tables to store row changes to tables at the source database that include user-defined types. Each column in the logging table must use a datatype that is supported by Streams in the source database release.
2. Before you begin the upgrade process described in "Performing a Database Upgrade Using Streams" on page B-6, create a DML trigger at the source database that fires on the tables that contain the user-defined datatypes. The trigger converts each row change into relational equivalents and logs the modified row in a logging table created in Step 1.
3. When the instructions in "Performing a Database Upgrade Using Streams" on page B-6 say to configure a capture process and **propagation**, configure the capture process and propagation to capture changes to the logging table and propagate these changes to the destination database. Changes to tables that contain user-defined types should not be captured or propagated.

4. When the instructions in "Performing a Database Upgrade Using Streams" on page B-6 say to configure a an apply process on the destination database, configure the apply process to use a **DML handler** that processes the changes to the logging tables. The DML handler reconstructs the user-defined types from the relational equivalents and applies the modified changes to the tables that contain user-defined types.

   See Also:

   - *Oracle Database Application Developer's Guide - Fundamentals* for more information about creating triggers
   - *Oracle Streams Replication Administrator's Guide* for more information about creating DML handlers

Deciding Which Utility to Use for Instantiation

Before you begin the database upgrade, decide whether you want to use the Export/Import utilities (Data Pump or original) or the Recovery Manager (RMAN) utility to instantiate the **destination database** during the operation. The destination database will replace the existing database that is being upgraded.

Consider the following factors when you make this decision:

- If you use original Export/Import or Data Pump Export/Import, then you can make the destination database an Oracle Database 10*g* Release 2 (10.2) database at the beginning of the operation. Therefore, you do not need to upgrade the destination database after the **instantiation**.

  If you use Export/Import for instantiation, and Data Pump is supported, then Oracle recommends using Data Pump. Data Pump can perform the instantiation faster than original Export/Import.

- If you use the RMAN `DUPLICATE` command, then the instantiation might be faster than with Export/Import, especially if the database is large, but the database release must be the same for RMAN instantiation. Therefore, if the database is an Oracle9*i* Database Release 2 (9.2) database, then the destination database is an Oracle9*i* Database Release 2 database when it is instantiated. If the database is an Oracle Database 10*g* Release 1 (10.1) database, then the destination database is an Oracle Database 10*g* Release 1 database when it is instantiated. After the instantiation, you must upgrade the destination database.

  Also, Oracle recommends that you do not use RMAN for instantiation in an environment where distributed transactions are possible. Doing so might cause in-doubt transactions that must be corrected manually.

Table B–2 describes whether each instantiation method is supported based on the release being upgraded, whether the platform at the source and destination databases are different, and whether the character set at the source and destination databases are different. Each instantiation method is supported when the platform and character set are the same at the source and destination databases.

Table B–2 Instantiation Methods for Database Upgrade with Streams

Instantiation Method	Supported When Upgrading From	Different Platforms Supported?	Different Character Sets Supported?
Original Export/Import	9.2 or 10.1	Yes	Yes
Data Pump Export/Import	10.1 only	Yes	Yes
RMAN `DUPLICATE`	9.2 or 10.1	No	No

Performing a Database Upgrade Using Streams

This section contains instructions for performing a database upgrade using Streams. To use Streams for a database upgrade, the database must be Oracle9*i* Database Release 2 (9.2) or later.

Complete the following tasks to upgrade a database using Streams:

- Task 1: Beginning the Upgrade
- Task 2: Setting Up Streams Prior to Instantiation
- Task 3: Instantiating the Database
- Task 4: Setting Up Streams After Instantiation
- Task 5: Finishing the Upgrade and Removing Streams

Task 1: Beginning the Upgrade

Complete the following steps to begin the upgrade using Oracle Streams:

1. Create an empty database. Make sure the **destination database** has a different global name than the source database. This example assumes that the global name of the source database is `orcl.net` and the global name of the destination database during the upgrade is `updb.net`. The global name of the destination database is changed when the destination database replaces the source database at the end of the upgrade process.

   The release of the empty database you create depends on the **instantiation** method you decided to use in "Deciding Which Utility to Use for Instantiation" on page B-5:

   - If you decided to use export/import for instantiation, then create an empty Oracle Database 10*g* Release 2 database. This database will be the destination database during the upgrade process.

     See the Oracle Database installation guide for your operating system if you need to install Oracle Database, and see the *Oracle Database Administrator's Guide* for information about creating a database.

   - If you decided to use RMAN for instantiation, then create an empty Oracle database that is the same release as the database you are upgrading.

     Specifically, if you are upgrading an Oracle9*i* Database Release 2 (9.2) database, then create an Oracle9*i* Release 2 database. Alternatively, if you are upgrading an Oracle Database 10*g* Release 1 (10.1) database, then create an Oracle Database 10*g* Release 1 database.

     This database will be the destination database during the upgrade process. Both the source database that is being upgraded and the destination database must be the same release of Oracle when you start the upgrade process.

See the Oracle installation guide for your operating system if you need to install Oracle, and see *Database Administrator's Guide* for the release for information about creating a database.

2. Make sure the source database is running in `ARCHIVELOG` mode. See the Oracle *Administrator's Guide* for the source database release for information about running a database in `ARCHIVELOG` mode.

3. Make sure the initialization parameters are set properly at each database to support a Streams environment. For the source database, see the Oracle Streams documentation for the source database release. For the destination database, see "Setting Initialization Parameters Relevant to Streams" on page 10-4. If the capture database is a third database, then see the Oracle Streams documentation for the capture database release.

4. At the source database, make read-only any database objects that were not supported by Oracle Streams in the release you are upgrading:

   - If you are upgrading an Oracle9*i* Database Release 2 (9.2) database, then make tables with columns of the following datatypes read-only: `NCLOB`, `LONG`, `LONG RAW`, `BFILE`, `ROWID`, and `UROWID`, and user-defined types (including object types, REFs, varrays, and nested tables). In addition, make the following types of tables read-only: temporary tables, index-organized tables, and object tables. See *Oracle9i Streams* for complete information about unsupported database objects.
   - If you are upgrading an Oracle Database 10*g* Release 1 (10.1) database, then query the `DBA_STREAMS_UNSUPPORTED` data dictionary view to list the database objects that are not supported by Streams. Make each of the listed database objects read-only.

   "Preparing to Upgrade a Database with User-defined Types" on page B-4 discusses a method for retaining changes to tables that contain user-defined types during the upgrade. If you are using this method, then tables that contain user-defined types can remain open during the upgrade.

5. At the source database, configure a Streams administrator:

   - If you are upgrading an Oracle9*i* Database Release 2 (9.2) database, then see *Oracle9i Streams* for instructions.
   - If you are upgrading an Oracle Database 10*g* Release 1 (10.1) database, then see the *Oracle Streams Concepts and Administration* book for that release for instructions.

   These instructions assume that the name of the Streams administrator at the source database is `strmadmin`. This Streams administrator will be copied automatically to the destination database during instantiation.

6. Connect as an administrative user in SQL*Plus to the source database, and specify **database supplemental logging** of primary keys, unique keys, and foreign keys for all updates. For example:

```
CONNECT SYSTEM/MANAGER@orcl.net

ALTER DATABASE ADD SUPPLEMENTAL LOG DATA
   (PRIMARY KEY, UNIQUE, FOREIGN KEY) COLUMNS;
```

Task 2: Setting Up Streams Prior to Instantiation

The specific instructions for setting up Streams prior to **instantiation** depend on which database is the capture database. Follow the instructions in the appropriate section:

- The Source Database Is the Capture Database
- The Destination Database Is the Capture Database
- A Third Database Is the Capture Database

> **See Also:** "Overview of Using Streams in the Database Upgrade Process" on page B-1 for information about the capture database

The Source Database Is the Capture Database

Complete the following steps to set up Streams prior to instantiation when the source database is the capture database:

1. Configure your network and Oracle Net so that the source database can communicate with the **destination database**. See *Oracle Database Net Services Administrator's Guide* for instructions.
2. Connect as the Streams administrator in SQL*Plus to the source database, and create an `ANYDATA` queue that will stage changes made to the source database during the upgrade process. For example:

```
CONNECT strmadmin/strmadminpw@orcl.net

BEGIN
  DBMS_STREAMS_ADM.SET_UP_QUEUE(
    queue_table => 'strmadmin.capture_queue_table',
    queue_name  => 'strmadmin.capture_queue');
END;
/
```

3. While still as the Streams administrator to the source database, configure a **capture process** that will capture all supported changes made to the source database and stage these changes in the **queue** created in Step 2. Do not start the capture process. For example:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_GLOBAL_RULES(
    streams_type       => 'capture',
    streams_name       => 'capture_upgrade',
    queue_name         => 'strmadmin.capture_queue',
    include_dml        => true,
    include_ddl        => true,
    include_tagged_lcr => false,
    source_database    => 'orcl.net',
    inclusion_rule     => true);
END;
/
```

   "Preparing to Upgrade a Database with User-defined Types" on page B-4 discusses a method for retaining changes to tables that contain user-defined types during the maintenance operation. If you are using this method, then make sure the capture process does not attempt to capture changes to tables with user-defined types. See the Streams documentation for the source database for information about excluding database objects from a Streams configuration with **rule**s.
4. Proceed to "Task 3: Instantiating the Database" on page B-11.

The Destination Database Is the Capture Database

The database being upgraded must be an Oracle Database 10*g* Release 1 database to use this option. Complete the following steps to set up Streams prior to instantiation when the **destination database** is the capture database:

1. Configure your network and Oracle Net so that the source database and destination database can communicate with each other. See *Oracle Database Net Services Administrator's Guide* for instructions.
2. Connect to the destination database as an administrative user, and create a Streams administrator. See "Configuring a Streams Administrator" on page 10-1 for instructions.

   These instructions assume that the name of the Streams administrator at the destination database is `strmadmin`.
3. Follow the instructions in the appropriate section based on the method you are using for instantiation:

   - Export/Import
   - RMAN

   Export/Import

   Complete the following steps if you are using export/import for instantiation:

   a. Connect as the Streams administrator in SQL*Plus to the destination database, and create an `ANYDATA` queue that will stage changes made to the source database during the upgrade process. For example:

   ```
   CONNECT strmadmin/strmadminpw@updb.net

   BEGIN
     DBMS_STREAMS_ADM.SET_UP_QUEUE(
       queue_table => 'strmadmin.destination_queue_table',
       queue_name  => 'strmadmin.destination_queue');
   END;
   /
   ```

   b. While still as the Streams administrator to the destination database, configure a downstream **capture process** that will capture all supported changes made to the source database and stage these changes in the **queue** created in Step a. Make sure the capture process uses a database link to the source database. The capture process can be a **real-time downstream capture process** or an **archived-log downstream capture process**. See "Creating a Capture Process" on page 11-2. Do not start the capture process.

   "Preparing to Upgrade a Database with User-defined Types" on page B-4 discusses a method for retaining changes to tables that contain user-defined types during the maintenance operation. If you are using this method, then make sure the capture process does not attempt to capture changes to tables with user-defined types. See the Streams documentation for the source database for information about excluding database objects from a Streams configuration with **rule**s.

RMAN

Complete the following steps if you are using RMAN for instantiation:

a. Connect as the Streams administrator in SQL*Plus to the source database, and perform a build of the data dictionary in the redo log:

```
CONNECT strmadmin/strmadminpw@orcl.net

SET SERVEROUTPUT ON
DECLARE
  scn  NUMBER;
BEGIN
  DBMS_CAPTURE_ADM.BUILD(
    first_scn => scn);
  DBMS_OUTPUT.PUT_LINE('First SCN Value = ' || scn);
END;
/
First SCN Value = 1122610
```

This procedure displays the valid **first SCN** value for the capture process that will be created at the destination database. Make a note of the SCN value returned because you will use it when you create the capture process at the destination database.

b. While still as the Streams administrator to the source database, prepare the source database for instantiation:

```
exec DBMS_CAPTURE_ADM.PREPARE_GLOBAL_INSTANTIATION();
```

4. Proceed to "Task 3: Instantiating the Database" on page B-11.

A Third Database Is the Capture Database

To use this option, meet the following requirements:

- The database being upgraded must be an Oracle Database 10*g* Release 1 database.
- The third database must be an Oracle Database 10*g* Release 1 or later database.

This example assumes that the global name of the third database is `thrd.net`. Complete the following steps to set up Streams prior to instantiation when a third database is the capture database:

1. Configure your network and Oracle Net so that the source database, **destination database**, and third database can communicate with each other. See *Oracle Database Net Services Administrator's Guide* for instructions.

2. Connect to the third database as an administrative user, and create a Streams administrator:

- If the third database is an Oracle Database 10*g* Release 1 database, then see the *Oracle Streams Concepts and Administration* book for that release for instructions.
- If the third database is an Oracle Database 10*g* Release 2 database, then see "Configuring a Streams Administrator" on page 10-1 for instructions.

These instructions assume that the name of the Streams administrator at the third database is `strmadmin`.

3. Connect as the Streams administrator in SQL*Plus to the third database, and create an ANYDATA queue that will stage changes made to the source database during the upgrade process. For example:

```
CONNECT strmadmin/strmadminpw@thrd.net

BEGIN
  DBMS_STREAMS_ADM.SET_UP_QUEUE(
    queue_table => 'strmadmin.capture_queue_table',
    queue_name  => 'strmadmin.capture_queue');
END;
/
```

4. While still connected as the Streams administrator to the third database, configure a downstream **capture process** that will capture all supported changes made to the source database and stage these changes in the **queue** created in Step 3. Make sure the capture process uses a database link to the source database. Do not start the capture process.

   See the following documentation for instructions:

   - If the capture database is an Oracle Database 10*g* Release 1 database, then see the *Oracle Streams Concepts and Administration* book for that release for instructions.
   - If the capture database is an Oracle Database 10*g* Release 2 database, then see "Creating a Capture Process" on page 11-2 for instructions. The capture process can be a **real-time downstream capture process** or an **archived-log downstream capture process**.

   "Preparing to Upgrade a Database with User-defined Types" on page B-4 discusses a method for retaining changes to tables that contain user-defined types during the maintenance operation. If you are using this method, then make sure the capture process does not attempt to capture changes to tables with user-defined types. See the Streams documentation for the source database for information about excluding database objects from a Streams configuration with **rule**s.

5. Proceed to "Task 3: Instantiating the Database" on page B-11.

Task 3: Instantiating the Database

"Deciding Which Utility to Use for Instantiation" on page B-5 discusses different options for instantiating an entire database. Complete the steps in the appropriate section based on the **instantiation** option you are using:

- Instantiating the Database Using Export/Import
- Instantiating the Database Using RMAN

Instantiating the Database Using Export/Import

Complete the following steps to instantiate the **destination database** using export/import:

1. Instantiate the destination database using Export/Import. See *Oracle Streams Replication Administrator's Guide* for more information about performing instantiations, and see *Oracle Database Utilities* for information about performing an export/import using the Export and Import utilities.

   If you use Oracle Data Pump or original Export/Import to instantiate the destination database, then make sure the following parameters are set to the appropriate values:

   - Set the `STREAMS_CONFIGURATION` import parameter to `n`.
   - If you use original Export/Import, then set the `CONSISTENT` export parameter to `y`. This parameter does not apply to Data Pump exports.
   - If you use original Export/Import, then set the `STREAMS_INSTANTIATION` import parameter to `y`. This parameter does not apply to Data Pump imports.

   If you are upgrading an Oracle9*i* Database Release 2 (9.2) database, then you must use original Export/Import.

2. At the destination database, disable any imported jobs that modify data that will be replicated from the source database. Query the `DBA_JOBS` data dictionary view to list the jobs.

3. Proceed to "Task 4: Setting Up Streams After Instantiation" on page B-15.

Instantiating the Database Using RMAN

Complete the following steps to instantiate the **destination database** using the RMAN `DUPLICATE` command:

> **Note:** These steps provide a general outline for using RMAN to duplicate a database. If you are upgrading an Oracle9*i* Release 2 database, then see the *Oracle9i Recovery Manager User's Guide* for detailed information about using RMAN in that release. If you upgrading an Oracle Database 10*g* Release 1 database, then see the *Oracle Database Backup and Recovery Advanced User's Guide* for that release.

1. Create a backup of the source database if one does not exist. RMAN requires a valid backup for duplication. In this example, create a backup of `orcl.net` if one does not exist.

2. While connected as an administrative user in SQL*Plus to the source database, determine the until SCN for the RMAN `DUPLICATE` command. For example:

```
CONNECT SYSTEM/MANAGER@orcl.net

SET SERVEROUTPUT ON SIZE 1000000
DECLARE
  until_scn NUMBER;
BEGIN
  until_scn:= DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER;
      DBMS_OUTPUT.PUT_LINE('Until SCN: ' || until_scn);
END;
/
```

Make a note of the until SCN value. This example assumes that the until SCN value is `439882`. You will set the `UNTIL SCN` option to this value when you use RMAN to duplicate the database in Step 5.

3. While connected as an administrative user in SQL*Plus to the source database, archive the current online redo log. For example:

```
CONNECT SYSTEM/MANAGER@orcl.net

ALTER SYSTEM ARCHIVE LOG CURRENT;
```

4. Prepare your environment for database duplication, which includes preparing the destination database as an auxiliary instance for duplication. See the documentation for the release from which you are upgrading for instructions. Specifically, see the "Duplicating a Database with Recovery Manager" chapter in the *Oracle9i Recovery Manager User's Guide* or *Oracle Database Backup and Recovery Advanced User's Guide* (10.1) for instructions.

5. Use the RMAN `DUPLICATE` command with the `OPEN RESTRICTED` option to instantiate the source database at the destination database. The `OPEN RESTRICTED` option is required. This option enables a restricted session in the duplicate database by issuing the following SQL statement: `ALTER SYSTEM ENABLE RESTRICTED SESSION`. RMAN issues this statement immediately before the duplicate database is opened.

   You can use the `UNTIL SCN` clause to specify an SCN for the duplication. Use the until SCN determined in Step 2 for this clause. Archived redo logs must be available for the until SCN specified and for higher SCN values. Therefore, Step 3 archived the redo log containing the until SCN.

   Make sure you use `TO` *`database_name`* in the `DUPLICATE` command to specify the name of the duplicate database. In this example, the duplicate database is `updb.net`. Therefore, the `DUPLICATE` command for this example includes `TO updb.net`.

   The following example is an RMAN `DUPLICATE` command:

```
rman
RMAN> CONNECT TARGET SYS/change_on_install@orcl.net
RMAN> CONNECT AUXILIARY SYS/change_on_install@updb.net
RMAN> RUN
      {
        SET UNTIL SCN 439882;
        ALLOCATE AUXILIARY CHANNEL updb DEVICE TYPE sbt;
        DUPLICATE TARGET DATABASE TO updb
        NOFILENAMECHECK
        OPEN RESTRICTED;
      }
```

6. While connected as an administrative user in SQL*Plus to the destination database, use the `ALTER SYSTEM` statement to disable the `RESTRICTED SESSION`:

```
CONNECT SYSTEM/MANAGER

ALTER SYSTEM DISABLE RESTRICTED SESSION;
```

7. While connected as an administrative user in SQL*Plus to the destination database, rename the database global name. After the RMAN `DUPLICATE` command, the destination database has the same global name as the source database, but the destination database must have its original name until the end of the upgrade. For example:

```
CONNECT SYSTEM/MANAGER

ALTER DATABASE RENAME GLOBAL_NAME TO updb.net;
```

8. At the destination database, disable any jobs that modify data that will be replicated from the source database. Query the `DBA_JOBS` data dictionary view to list the jobs.
9. Upgrade the destination database to Oracle Database 10*g* Release 2. See the *Oracle Database Upgrade Guide* for instructions.
10. If you have not done so already, configure your network and Oracle Net so that the source database and destination database can communicate with each other. See *Oracle Database Net Services Administrator's Guide* for instructions.
11. Connect as the Streams administrator in SQL*Plus to the destination database, and create a database link to the source database. For example:

```
CONNECT strmadmin/strmadminpw@updb.net

CREATE DATABASE LINK orcl.net CONNECT TO strmadmin IDENTIFIED BY strmadminpw
   USING 'orcl.net';
```

12. While connected as the Streams administrator in SQL*Plus to the destination database, set the **instantiation SCN** for the entire database and all of the database objects. The RMAN `DUPLICATE` command duplicates the database up to one less than the SCN value specified in the `UNTIL SCN` clause. Therefore, you should subtract one from the until SCN value that you specified when you ran the `DUPLICATE` command in Step 5. In this example, the until SCN was set to `439882`. Therefore, the instantiation SCN should be set to `439882 - 1`, or `439881`.

```
CONNECT strmadmin/strmadminpw@updb.net

BEGIN
  DBMS_APPLY_ADM.SET_GLOBAL_INSTANTIATION_SCN(
    source_database_name => 'orcl.net',
    instantiation_scn    => 439881,
    recursive            => true);
END;
/
```

13. Proceed to "Task 4: Setting Up Streams After Instantiation" on page B-15.

Task 4: Setting Up Streams After Instantiation

The specific instructions for setting up Streams after **instantiation** depend on which database is the capture database. Follow the instructions in the appropriate section:

- The Source Database Is the Capture Database
- The Destination Database Is the Capture Database
- A Third Database Is the Capture Database

> **See Also:** "Overview of Using Streams in the Database Upgrade Process" on page B-1 for information about the capture database

The Source Database Is the Capture Database

Complete the following steps to set up Streams after instantiation when the source database is the capture database:

1. Connect as the Streams administrator in SQL*Plus to the **destination database**, and remove the Streams components that were cloned from the source database during instantiation:

   - If export/import was used for instantiation, then remove the `ANYDATA` queue that was cloned from the source database.
   - If RMAN was used for instantiation, then remove the `ANYDATA` queue and the capture process that were cloned from the source database.

   To remove the **queue** that was cloned from the source database, run the `REMOVE_QUEUE` procedure in the `DBMS_STREAMS_ADM` package. For example:

```
CONNECT strmadmin/strmadminpw@updb.net

BEGIN
  DBMS_STREAMS_ADM.REMOVE_QUEUE(
    queue_name              => 'strmadmin.capture_queue',
    cascade                 => false,
    drop_unused_queue_table => true);
END;
/
```

   To remove the capture process that was cloned from the source database, run the `DROP_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package. For example:

```
CONNECT strmadmin/strmadminpw@updb.net

BEGIN
  DBMS_CAPTURE_ADM.DROP_CAPTURE(
    capture_name          => 'capture_upgrade',
    drop_unused_rule_sets => true);
END;
/
```

2. While connected as the Streams administrator in SQL*Plus to the destination database, create an ANYDATA queue. This queue will stage changes propagated from the source database. For example:

```
CONNECT strmadmin/strmadminpw@updb.net

BEGIN
  DBMS_STREAMS_ADM.SET_UP_QUEUE(
    queue_table => 'strmadmin.destination_queue_table',
    queue_name  => 'strmadmin.destination_queue');
END;
/
```

3. Connect as the Streams administrator in SQL*Plus to the source database, and create a database link to the destination database. For example:

```
CONNECT strmadmin/strmadminpw@orcl.net

CREATE DATABASE LINK updb.net CONNECT TO strmadmin IDENTIFIED BY strmadminpw
   USING 'updb.net';
```

4. While connected as the Streams administrator in SQL*Plus to the source database, create a **propagation** that propagates all changes from the **source queue** to the **destination queue** created in Step 2. For example:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_GLOBAL_PROPAGATION_RULES(
    streams_name            => 'to_updb',
    source_queue_name       => 'strmadmin.capture_queue',
    destination_queue_name  => 'strmadmin.destination_queue@updb.net',
    include_dml             => true,
    include_ddl             => true,
    include_tagged_lcr      => true,
    source_database         => 'orcl.net');
END;
/
```

5. Connect as the Streams administrator in SQL*Plus to destination database, and create an **apply process** that applies all changes in the queue created in Step 2. For example:

```
CONNECT strmadmin/strmadminpw@updb.net

BEGIN
  DBMS_STREAMS_ADM.ADD_GLOBAL_RULES(
    streams_type       => 'apply',
    streams_name       => 'apply_upgrade',
    queue_name         => 'strmadmin.destination_queue',
    include_dml        => true,
    include_ddl        => true,
    include_tagged_lcr => true,
    source_database    => 'orcl.net');
END;
/
```

6. Proceed to "Task 5: Finishing the Upgrade and Removing Streams" on page B-19.

The Destination Database Is the Capture Database

Complete the following steps to set up Streams after instantiation when the **destination database** is the capture database:

1. Complete the following steps if you used RMAN for instantiation. If you used export/import for instantiation, then proceed to Step 2.

   a. Connect as the Streams administrator in SQL*Plus to the destination database, and create an ANYDATA queue that will stage changes made to the source database during the upgrade process. For example:

   ```
   CONNECT strmadmin/strmadminpw@updb.net

   BEGIN
     DBMS_STREAMS_ADM.SET_UP_QUEUE(
       queue_table => 'strmadmin.destination_queue_table',
       queue_name  => 'strmadmin.destination_queue');
   END;
   /
   ```

   b. While still as the Streams administrator to the destination database, configure a **downstream capture process** that will capture all supported changes made to the source database and stage these changes in the **queue** created in Step a.

   Make sure you set the `first_scn` parameter in the `CREATE_CAPTURE` procedure to the value obtained for the data dictionary build in Step 3a on page B-10 in "The Destination Database Is the Capture Database". In this example, the `first_scn` parameter should be set to `1122610`.

   The capture process can be a **real-time downstream capture process** or an **archived-log downstream capture process**. See "Creating a Capture Process" on page 11-2. Do not start the capture process.

   "Preparing to Upgrade a Database with User-defined Types" on page B-4 discusses a method for retaining changes to tables that contain user-defined types during the maintenance operation. If you are using this method, then make sure the capture process does not attempt to capture changes to tables with user-defined types. See the Streams documentation for the source database for information about excluding database objects from a Streams configuration with **rule**s.

2. Connect as the Streams administrator in SQL*Plus to destination database, and create an **apply process** that applies all changes in the queue used by the downstream capture process. For example:

   ```
   CONNECT strmadmin/strmadminpw@updb.net

   BEGIN
     DBMS_STREAMS_ADM.ADD_GLOBAL_RULES(
       streams_type       => 'apply',
       streams_name       => 'apply_upgrade',
       queue_name         => 'strmadmin.destination_queue',
       include_dml        => true,
       include_ddl        => true,
       include_tagged_lcr => true,
       source_database    => 'orcl.net');
   END;
   /
   ```

3. Proceed to "Task 5: Finishing the Upgrade and Removing Streams" on page B-19.

A Third Database Is the Capture Database

This example assumes that the global name of the third database is `thrd.net`. Complete the following steps to set up Streams after instantiation when a third database is the capture database:

1. Connect as the Streams administrator in SQL*Plus to the **destination database**, and create an `ANYDATA` queue. This **queue** will stage changes propagated from the capture database. For example:

```
CONNECT strmadmin/strmadminpw@updb.net

BEGIN
  DBMS_STREAMS_ADM.SET_UP_QUEUE(
    queue_table => 'strmadmin.destination_queue_table',
    queue_name  => 'strmadmin.destination_queue');
END;
/
```

2. Connect as the Streams administrator in SQL*Plus to the capture database, and create a database link to the destination database. For example:

```
CONNECT strmadmin/strmadminpw@thrd.net

CREATE DATABASE LINK updb.net CONNECT TO strmadmin IDENTIFIED BY strmadminpw
   USING 'updb.net';
```

3. While connected as the Streams administrator in SQL*Plus to the capture database, create a **propagation** that propagates all changes from the **source queue** at the capture database to the **destination queue** created in Step 1. For example:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_GLOBAL_PROPAGATION_RULES(
    streams_name            => 'to_updb',
    source_queue_name       => 'strmadmin.capture_queue',
    destination_queue_name  => 'strmadmin.destination_queue@updb.net',
    include_dml             => true,
    include_ddl             => true,
    include_tagged_lcr      => true,
    source_database         => 'orcl.net');
END;
/
```

4. Connect as the Streams administrator in SQL*Plus to destination database, and create an **apply process** that applies all changes in the queue created in Step 1. For example:

```
CONNECT strmadmin/strmadminpw@updb.net

BEGIN
  DBMS_STREAMS_ADM.ADD_GLOBAL_RULES(
    streams_type       => 'apply',
    streams_name       => 'apply_upgrade',
    queue_name         => 'strmadmin.destination_queue',
    include_dml        => true,
    include_ddl        => true,
    include_tagged_lcr => true,
    source_database    => 'orcl.net');
END;
/
```

5. Complete the steps in "Task 5: Finishing the Upgrade and Removing Streams" on page B-19.

Task 5: Finishing the Upgrade and Removing Streams

Complete the following steps to finish the upgrade operation using Oracle Streams and remove Streams components:

1. Connect as the Streams administrator in SQL*Plus to the **destination database**, and start the apply process. For example:

```
CONNECT strmadmin/strmadminpw@updb.net

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

2. Connect as the Streams administrator in SQL*Plus to the capture database, and start the capture process. For example:

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

   This step begins the process of replicating changes that were made to the source database during instantiation of the destination database.

3. While connected as the Streams administrator in SQL*Plus to the capture database, monitor the Streams environment until the apply process at the destination database has applied most of the changes from the source database. For example, if the name of the capture process is `capture_upgrade`, and the name of the apply process is `apply_upgrade`, then run the following query at the source database:

```
COLUMN ENQUEUE_MESSAGE_NUMBER HEADING 'Captured SCN' FORMAT 99999999999
COLUMN LWM_MESSAGE_NUMBER HEADING 'Applied SCN' FORMAT 99999999999

SELECT c.ENQUEUE_MESSAGE_NUMBER, a.LWM_MESSAGE_NUMBER
  FROM V$STREAMS_CAPTURE c, V$STREAMS_APPLY_COORDINATOR@updb.net a
  WHERE CAPTURE_NAME = 'CAPTURE_UPGRADE'
    AND APPLY_NAME   = 'APPLY_UPGRADE';
```

   When the two SCN values returned by this query are nearly equal, most of the changes from the source database have been applied at the destination database, and you can proceed to the next step. At this point in the process, the values returned by this query might never be equal because the source database still allows changes.

   If this query returns no results, then make sure the **Streams client**s in the environment are enabled by querying the `STATUS` column in the `DBA_CAPTURE` view at the capture database and the `DBA_APPLY` view at the destination database. If a **propagation** is used, you can check the status of the propagation by running the query in "Displaying the Schedule for a Propagation Job" on page 21-16.

   If a Streams client is disabled, then try restarting it. If a Streams client will not restart, then troubleshoot the environment using the information in Chapter 18, "Troubleshooting a Streams Environment".

4. Connect as the Streams administrator in SQL*Plus to the destination database, and make sure there are no apply errors by running the following query:

```
CONNECT strmadmin/strmadminpw@updb.net

SELECT COUNT(*) FROM DBA_APPLY_ERROR;
```

   If this query returns zero, then proceed to the next step. If this query shows errors in the error queue, then resolve these errors before continuing. See "Managing Apply Errors" on page 13-23 for instructions.

5. Disconnect all applications and users from the source database.

6. Connect as an administrative user in SQL*Plus to the source database, and restrict access to the database. For example:

```
CONNECT SYSTEM/MANAGER@orcl.net

ALTER SYSTEM ENABLE RESTRICTED SESSION;
```

7. Connect as an administrative user in SQL*Plus to the capture database, and repeat the query you ran in Step 3. When the two SCN values returned by the query are equal, all of the changes from the source database have been applied at the destination database, and you can proceed to the next step.

8. Connect as the Streams administrator in SQL*Plus to the destination database, and repeat the query you ran in Step 4. If this query returns zero, then move on to the next step. If this query shows errors in the error queue, then resolve these errors before continuing. See "Managing Apply Errors" on page 13-23 for instructions.

9. If you performed any actions that created, modified, or deleted job queue processes at the source database during the upgrade process, then perform the same actions at the destination database. See "Considerations for Job Queue Processes and PL/SQL Package Subprograms" on page B-4 for more information.

10. If you invoked any Oracle-supplied PL/SQL package subprograms at the source database during the upgrade process that modified both user data and dictionary metadata at the same time, then invoke the same subprograms at the destination database. See "Considerations for Job Queue Processes and PL/SQL Package Subprograms" on page B-4 for more information.

11. Shut down the source database. This database should not be opened again.

12. While connected as an administrative user in SQL*Plus to the destination database, change the global name of the database to match the source database. For example:

```
ALTER DATABASE RENAME GLOBAL_NAME TO orcl.net;
```

13. At the destination database, enable any jobs that you disabled earlier.

14. Make the destination database available for applications and users. Redirect any applications and users that were connecting to the source database to the destination database. If necessary, reconfigure your network and Oracle Net so that systems that communicated with the source database now communicate with the destination database. See *Oracle Database Net Services Administrator's Guide* for instructions.

15. At the destination database, remove the Streams components that are no longer needed. Connect as an administrator with SYSDBA privilege to the destination database, and run the following procedure:

    Note: Running this procedure is dangerous. It removes the local Streams configuration. Make sure you are ready to remove the Streams configuration at the destination database before running this procedure.

    ```
    EXEC DBMS_STREAMS_ADM.REMOVE_STREAMS_CONFIGURATION();
    ```

    If you no longer need **database supplemental logging** at the destination database, then run the following statement to drop it:

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

    If you no longer need the Streams administrator at the destination database, then run the following statement:

    ```
    DROP USER strmadmin CASCADE;
    ```

16. If the capture database was a third database, then, at the third database, remove the Streams components that are no longer needed. Connect as an administrator with SYSDBA privilege to the third database, and run the following procedure:

    Note: Running this procedure is dangerous. It removes the local Streams configuration. Make sure you are ready to remove the Streams configuration at the third database before running this procedure.

    ```
    EXEC DBMS_STREAMS_ADM.REMOVE_STREAMS_CONFIGURATION();
    ```

    If you no longer need database **supplemental logging** at the third database, then run the following statement to drop it:

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

    If you no longer need the Streams administrator at the destination database, then run the following statement:

    ```
    DROP USER strmadmin CASCADE;
    ```

The database upgrade is complete.

C

Online Database Maintenance with Streams

This appendix describes how to perform some maintenance operations with Oracle Streams on an Oracle Database 10*g* Release 2 database. These maintenance operations include migrating an Oracle Database to a different platform or character set, upgrading user-created applications, and applying Oracle Database patches. The maintenance operations described in this appendix use the features of Oracle Streams to achieve little or no database down time.

This appendix contains these topics:

- Overview of Using Streams for Database Maintenance Operations
- Preparing for a Database Maintenance Operation
- Performing a Database Maintenance Operation Using Streams

> **See Also:** Appendix B, "Online Database Upgrade with Streams" for instructions on performing an upgrade of a prior release of Oracle Database with Streams

Overview of Using Streams for Database Maintenance Operations

The following maintenance operations typically require substantial database down time:

- Migrating a database to a different platform
- Migrating a database to a different character set
- Modifying database schema objects to support upgrades to user-created applications
- Applying an Oracle Database software patch

You can achieve these maintenance operations with little or no down time by using the features of Oracle Streams. To do so, you use Oracle Streams to configure a single-source **replication** environment with the following databases:

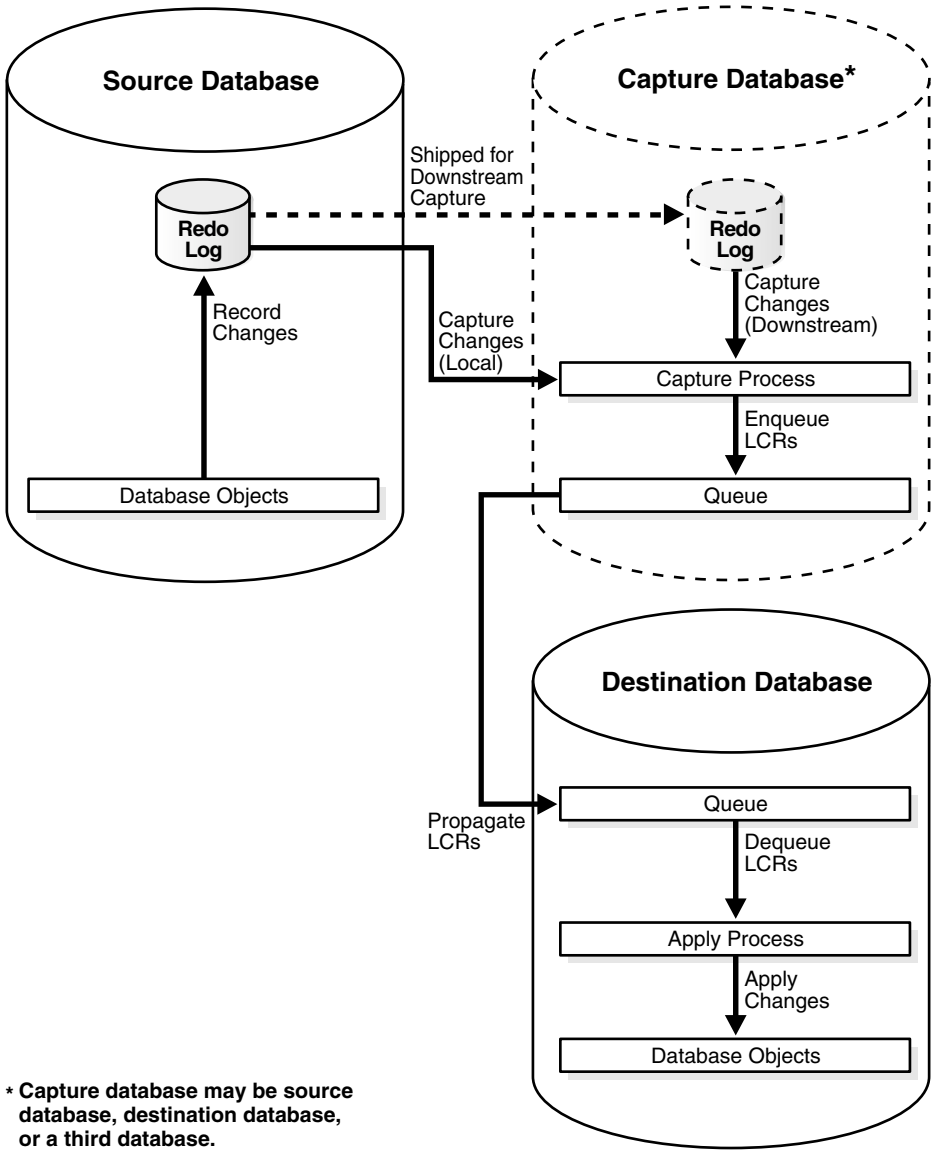
- **Source Database:** The original database that is being maintained.
- **Capture Database:** The database where a **capture process** captures changes made to the source database during the maintenance operation.
- **Destination Database:** The copy of the source database where an **apply process** applies changes made to the source database during the maintenance operation.

Specifically, you can use the following general steps to perform the maintenance operation while the database is online:

1. Create an empty **destination database**.
2. Configure an Oracle Streams single-source replication environment where the original database is the source database and a copy of the database is the destination database. The `PRE_INSTANTIATION_SETUP` and `POST_INSTANTIATION_SETUP` procedures in the `DBMS_STREAMS_ADM` package configure the Streams replication environment.
3. Perform the maintenance operation on the destination database. During this time the original source database is available online, and changes to the original source database are being captured by a capture process.
4. Use Oracle Streams to apply the changes made to the source database at the destination database.
5. When the destination database has caught up with the changes made at the source database, take the source database offline and make the destination database available for applications and users.

Figure C–1 provides an overview of this process.

Figure C–1 Online Database Maintenance with Streams



The Capture Database During the Maintenance Operation

During the maintenance operation, the capture database is the database where the **capture process** is created. A **local capture process** can be created at the source database during the maintenance operation, or a **downstream capture process** can be created at the **destination database** or at a third database. If the destination database is the capture database, then a **propagation** from the capture database to the destination database is not needed. A downstream capture process reduces the resources required at the source database during the maintenance operation.

Note:

- Before you begin the database maintenance operation with Streams, decide which database will be the capture database.
- If the RMAN `DUPLICATE` or `CONVERT DATABASE` command is used for database **instantiation**, then the destination database cannot be the capture database.

See Also:

- "Local Capture and Downstream Capture" on page 2-12
- "Deciding Which Utility to Use for Instantiation" on page C-11

Assumptions for the Database Being Maintained

The instructions in this appendix assume that all of the following statements are true for the database being maintained:

- The database is not part of an existing Oracle Streams environment.
- The database is not part of an existing logical standby environment.
- The database is not part of an existing Advanced Replication environment.
- No tables at the database are master tables for materialized views in other databases.
- Any user-created **queue**s are read-only during the maintenance operation.

Considerations for Job Queue Processes and PL/SQL Package Subprograms

If possible, ensure that no job queue processes are created, modified, or deleted during the maintenance operation, and that no Oracle-supplied PL/SQL package subprograms are invoked during the maintenance operation that modify both user data and data dictionary metadata at the same time. The following packages contain subprograms that modify both user data and data dictionary metadata at the same time: `DBMS_RLS`, `DBMS_STATS`, and `DBMS_JOB`.

It might be possible to perform such actions on the database if you ensure that the same actions are performed on the source database and **destination database** in Steps 13 and 14 in "Performing a Database Maintenance Operation Using Streams" on page C-12. For example, if a PL/SQL procedure gathers statistics on the source database during the maintenance operation, then the same PL/SQL procedure should be invoked at the destination database in Step 14.

Unsupported Database Objects Are Excluded

The `PRE_INSTANTIATION_SETUP` and `POST_INSTANTIATION_SETUP` procedures in the `DBMS_STREAMS_ADM` package include the following parameters:

- `exclude_schemas`
- `exclude_flags`

These parameters specify which database objects to exclude from the Streams configuration. The examples in this appendix set these parameters to the following values:

```
exclude_schemas => '*',
exclude_flags   => DBMS_STREAMS_ADM.EXCLUDE_FLAGS_UNSUPPORTED +
                   DBMS_STREAMS_ADM.EXCLUDE_FLAGS_DML +
                   DBMS_STREAMS_ADM.EXCLUDE_FLAGS_DDL);
```

These values exclude any database objects that are not supported by Streams. The asterisk (`*`) specified for `exclude_schemas` indicates that some database objects in every schema in the database might be excluded from the **replication** environment. The value specified for the `exclude_flags` parameter indicates that DML and DDL changes for all unsupported database objects are excluded from the replication environment. Rules are placed in the **negative rule set**s for the **capture process**es to exclude these database objects.

To list unsupported database objects, query the `DBA_STREAMS_UNSUPPORTED` data dictionary view at the source database. If you use these parameter settings, then changes to the database objects listed in this view are not maintained by Streams during the maintenance operation. Therefore, Step 6 on page C-13 in "Task 1: Beginning the Maintenance Operation" instructs you to make these database objects read-only during the database maintenance operation.

Note: "Preparing for Maintenance of a Database with User-defined Types" on page C-7 discusses a method for retaining changes to tables that contain user-defined types during the maintenance operation. If you are using this method, then tables that contain user-defined types can remain open during the maintenance operation.

See Also: *Oracle Database PL/SQL Packages and Types Reference* for more information about the `exclude_schemas` and `exclude_flags` parameters

Preparing for a Database Maintenance Operation

The following sections describe tasks to complete before starting the database maintenance operation with Streams:

- Preparing for Downstream Capture
- Preparing for Maintenance of a Database with User-defined Types
- Preparing for Upgrades to User-Created Applications
- Deciding Whether to Configure Streams Directly or Generate a Script
- Deciding Which Utility to Use for Instantiation

Preparing for Downstream Capture

If you decided that the **destination database** or a third database will be the capture database, then you must prepare for downstream capture by configuring log file copying from the source database to the capture database. If you decided that the source database will be the capture database, then log file copying is not required.

Complete the following steps to prepare the source database to copy its redo log files to the capture database, and to prepare the capture database to accept these redo log files:

1. Configure Oracle Net so that the source database can communicate with the capture database.

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

2. Set the following initialization parameters to configure redo transport services to copy archived redo log files from the source database to the capture database:

   - At the source database, set at least one archive log destination in the `LOG_ARCHIVE_DEST_n` initialization parameter to a directory on the computer system running the capture database. To do this, set the following attributes of this parameter:
     - `SERVICE` - Specify the network service name of the capture database.
     - `ARCH` or `LGWR ASYNC` - If you specify `ARCH` (the default), then the archiver process (`ARCn`) will archive the redo log files to the capture database. If you specify `LGWR ASYNC`, then the log writer process (`LGWR`) will archive the redo log files to the capture database. Either `ARCH` or `LGWR ASYNC` is acceptable for a capture database destination.
     - `MANDATORY` or `OPTIONAL` - If you specify `MANDATORY`, then archiving of a redo log file to the capture database must succeed before the corresponding online redo log at the source database can be overwritten. If you specify `OPTIONAL`, then successful archiving of a redo log file to the capture database is not required before the corresponding online redo log at the source database can be overwritten. Either `MANDATORY` or `OPTIONAL` is acceptable for a capture database destination. If neither the `MANDATORY` nor the `OPTIONAL` attribute is specified, then the default is `OPTIONAL`.
     - `NOREGISTER` - Specify this attribute so that the capture database location is not recorded in the capture database control file.
     - `TEMPLATE` - Specify a directory and format template for archived redo logs at the capture database. The `TEMPLATE` attribute overrides the `LOG_ARCHIVE_FORMAT` initialization parameter settings at the capture database. The `TEMPLATE` attribute is valid only with remote destinations. Make sure the format uses all of the following variables at each source database: `%t`, `%s`, and `%r`.

     The following is an example of an `LOG_ARCHIVE_DEST_n` setting that specifies a capture database (`DBS2.NET`):

```
LOG_ARCHIVE_DEST_2='SERVICE=DBS2.NET ARCH OPTIONAL NOREGISTER
   TEMPLATE=/usr/oracle/log_for_dbs1/dbs1_arch_%t_%s_%r.log'
```

If another source database transfers log files to this capture database, then, in the initialization parameter file at this other source database, you can use the TEMPLATE attribute to specify a different directory and format for the log files at the capture database. The log files from each source database are kept separate at the capture database.

Tip: Log files from a remote source database should be kept separate from local database log files. In addition, if the capture database contains log files from multiple source databases, then the log files from each source database should be kept separate from each other.

- At the source database, set the LOG_ARCHIVE_DEST_STATE_*n* initialization parameter that corresponds with the LOG_ARCHIVE_DEST_*n* parameter for the capture database to ENABLE.

  For example, if the LOG_ARCHIVE_DEST_2 initialization parameter is set for the capture database, then set one LOG_ARCHIVE_DEST_STATE_2 parameter in the following way:

  ```
  LOG_ARCHIVE_DEST_STATE_2=ENABLE
  ```

- At the source database, make sure the setting for the LOG_ARCHIVE_CONFIG initialization parameter includes the send value.
- At the downstream database, make sure the setting for the LOG_ARCHIVE_CONFIG initialization parameter includes the receive value.

  See Also: *Oracle Database Reference* and *Oracle Data Guard Concepts and Administration* for more information about these initialization parameters

3. If you reset any initialization parameters while the instance is running at a database in Step 2, then you might want to reset them in the initialization parameter file as well, so that the new values are retained when the database is restarted.

   If you did not reset the initialization parameters while the instance was running, but instead reset them in the initialization parameter file in Step 2, then restart the database. The source database must be open when it sends redo log files to the capture database because the global name of the source database is sent to the capture database only if the source database is open.

   See Also: "Overview of Using Streams for Database Maintenance Operations" on page C-2 for more information about the capture database

Preparing for Maintenance of a Database with User-defined Types

User-defined types include object types, REF values, varrays, and nested tables. Currently, Streams **capture process**es and **apply process**es do not support user-defined types. This section discusses using Streams to perform a maintenance operation on a database that has user-defined types.

One option is to make tables that contain user-defined types read-only during the database maintenance operation. In this case, these tables are instantiated at the **destination database**, and no changes are made to these tables during the entire operation. After the maintenance operation is complete, make the tables that contain user-defined types read/write at the destination database.

However, if tables that contain user-defined types must remain open during the maintenance operation, then the following general steps can be used to retain changes to these types during the database maintenance operation:

1. At the source database, create one or more logging tables to store row changes to tables that include user-defined types. Each column in the logging table must use a datatype that is supported by Streams.
2. At the source database, create a DML trigger that fires on the tables that contain the user-defined datatypes. The trigger converts each row change into relational equivalents and logs the modified row in a logging table created in Step 1.
3. Make sure the capture process and **propagation** are configured to capture and, if necessary, propagate changes made to the logging table to the destination database. Changes to tables that contain user-defined types should not be captured or propagated. Therefore, make sure the `PRE_INSTANTIATION_SETUP` and `POST_INSTANTIATION_SETUP` procedures include the logging tables and exclude the tables that contain user-defined types.
4. At the destination, configure the apply process to use a **DML handler** that processes the changes to the logging tables. The DML handler reconstructs the user-defined types from the relational equivalents and applies the modified changes to the tables that contain user-defined types.

See Also:

- *Oracle Database Application Developer's Guide - Fundamentals* for more information about creating triggers
- *Oracle Streams Replication Administrator's Guide* for more information about creating DML handlers

Preparing for Upgrades to User-Created Applications

This section is relevant only if the maintenance operation entails upgrading user-created applications. During an upgrade of user-created applications, schema objects can be modified, and there might be logical dependencies that cannot be detected by the database alone. The following sections describe handling these issues during an application upgrade:

- Handling Modifications to Schema Objects
- Handling Logical Dependencies

Handling Modifications to Schema Objects

If you are upgrading user-created applications, then, typically, schema objects in the database change to support the upgraded applications. In Streams, row LCRs contain information about row changes that result from DML statements. A **declarative rule-based transformation** or **DML handler** can modify row LCRs captured from the source database redo log so that the row LCRs can be applied to the altered tables at the **destination database**.

A **rule-based transformation** is any modification to a **message** that results when a **rule** in a **positive rule set** evaluates to `TRUE`. Declarative rule-based transformation cover one of a common set of transformation scenarios for row LCRs. Declarative rule-based transformations are run internally without using PL/SQL. You specify such a transformation using a procedure in the `DBMS_STREAMS_ADM` package. A declarative rule-based transformation can modify row LCRs during capture, propagation, or apply.

A **DML handler** is a user procedure that processes row LCRs resulting from DML statements at a source database. A Streams **apply process** at a destination database can pass row LCRs to a DML handler, and the DML handler can modify the row LCRs.

The process for upgrading user-created applications using Streams can involve modifying and creating the schema objects at the destination database after **instantiation**. You can use one or more declarative rule-based transformations and DML handlers at the destination database to process changes from the source database so that they apply to the modified schema objects correctly. Declarative rule-based transformations and DML handlers can be used during application upgrade to account for differences between the source database and destination database.

In general, declarative rule-based transformations are easier to use than DML handlers. Therefore, when modifications to row LCRs are required, try to configure a declarative rule-based transformation first. If a declarative rule-based transformation is not sufficient, then use a DML handler. If row LCRs for tables that contain one or more LOB columns must be modified, then you should use a DML handler and **LOB assembly**.

Before you begin the database maintenance operation, you should complete the following tasks to prepare your declarative rule-based transformations or DML handlers:

- Learn about declarative rule-based transformations. See "Declarative Rule-Based Transformations" on page 7-1.
- Learn about DML handlers. See "Message Processing Options for an Apply Process" on page 4-3.
- Determine the declarative rule-based transformations and DML handlers you will need at your destination database. Your determination depends on the modifications to the schema objects required by your upgraded applications.
- Create the PL/SQL procedures that you will use for any DML handlers during the database maintenance operation. See the *Oracle Streams Replication Administrator's Guide* for information about creating the PL/SQL procedures.
- If row LCRs for tables that contain one or more LOB columns must be modified, then learn about using LOB assembly. See *Oracle Streams Replication Administrator's Guide*.

> **Note:** Custom rule-based transformation can also be used to modify row LCRs during application upgrade. However, these modifications can be accomplished using DML handlers, and DML handlers are more efficient than **custom rule-based transformation**s.

Handling Logical Dependencies

In some cases, an **apply process** requires additional information to detect dependencies in row LCRs that are being applied in parallel. During application upgrades, an apply process might require additional information to detect dependencies in the following situations:

- The application, rather than the database, enforces logical dependencies.
- Schema objects have been modified to support the application upgrade, and a **DML handler** will modify row LCRs to account for differences between the source database and destination database.

A **virtual dependency definition** is a description of a dependency that is used by an apply process to detect dependencies between transactions at a destination database. A virtual dependency definition is not described as a constraint in the destination database data dictionary. Instead, it is specified using procedures in the DBMS_APPLY_ADM package. Virtual dependency definitions enable an apply process to detect dependencies that it would not be able to detect by using only the constraint information in the data dictionary. After dependencies are detected, an apply process schedules LCRs and transactions in the correct order for apply.

If virtual dependency definitions are required for your application upgrade, then learn about virtual dependency definitions and plan to configure them during the application upgrade.

> **See Also:** *Oracle Streams Replication Administrator's Guide* for more information about virtual dependency definitions

Deciding Whether to Configure Streams Directly or Generate a Script

The PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP procedures in the DBMS_STREAMS_ADM package configure the Streams **replication** environment during the maintenance operation. These procedures can configure the Streams replication environment directly, or they can generate a script that configures the environment. Using a procedure to configure replication directly is simpler than running a script, and the environment is configured immediately. However, you might choose to generate a script for the following reasons:

- You want to review the actions performed by the procedure before configuring the environment.
- You want to modify the script to customize the configuration.

To configure Streams directly when you run one of these procedures, set the perform_actions parameter to true. The examples in this appendix assume that the procedures will configure Streams directly.

To generate a configuration script when you run one of these procedures, complete the following steps when you are instructed to run a procedure in this appendix:

1. In SQL*Plus, connect as the Streams administrator to database where you will run the procedure, and create a directory object to store the script that will be generated by the procedure. For example:

```
CONNECT strmadmin/strmadminpw

CREATE DIRECTORY scripts_dir AS '/usr/scripts';
```

2. While still connected to the source database as the Streams administrator, run the procedure. Make sure the following parameters are set to generate a script:
   - Set the perform_actions parameter to false.
   - Set the script_name parameter to the name of the script you want to generate.
   - Set the script_directory_object parameter to the directory object into which you want to place the script.
3. Review or modify the script, if necessary.

4. In SQL*Plus, connect as the Streams administrator, and run the generated script. For example:

```
CONNECT strmadmin/strmadminpw

@/usr/scripts/pre_instantiation.sql;
```

Deciding Which Utility to Use for Instantiation

Before you begin the database maintenance operation, decide whether you want to use Export/Import utilities (Data Pump or original) or the Recovery Manager (RMAN) utility to instantiate the **destination database** during the operation. Consider the following factors when you make this decision:

- If you are migrating the database to a different platform, then you can use either Export/Import or the RMAN `CONVERT DATABASE` command. The RMAN `DUPLICATE` command does not support migrating a database to a different platform.
- If you are migrating the database to a different character set, then you must use Export/Import. The RMAN `DUPLICATE` and `CONVERT DATABASE` commands do not support migrating a database to a different character set.
- If RMAN is supported for the operation, then using RMAN for the **instantiation** might be faster than using Export/Import, especially if the database is large.
- Oracle recommends that you do not use RMAN for instantiation in an environment where distributed transactions are possible. Doing so might cause in-doubt transactions that must be corrected manually.
- If you use Export/Import for instantiation, then Oracle recommends using Data Pump. Data Pump typically performs the instantiation faster than original Export/Import.
- If the RMAN `DUPLICATE` or `CONVERT DATABASE` command is used for database instantiation, then the destination database cannot be the capture database.

Table C–1 describes when each instantiation method is supported based on whether the platform at the source and destination databases are the same or different, and whether the character set at the source and destination databases are the same or different.

Table C–1 Instantiation Methods for Database Maintenance with Streams

Instantiation Method	Same Platform Supported?	Different Platforms Supported?	Same Character Set Supported?	Different Character Sets Supported?
Original Export/Import	Yes	Yes	Yes	Yes
Data Pump Export/Import	Yes	Yes	Yes	Yes
RMAN `DUPLICATE`	Yes	No	Yes	No
RMAN `CONVERT DATABASE`	No	Maybe	Yes	No

Only some platform combinations are supported by the RMAN `CONVERT DATABASE` command. You can use the `DBMS_TDB` package to determine whether a platform combination is supported.

See Also:

- *Oracle Streams Replication Administrator's Guide* for more information about Streams instantiations
- *Oracle Database Backup and Recovery Advanced User's Guide* for instructions on using the RMAN `DUPLICATE` and `CONVERT DATABASE` commands
- *Oracle Database Backup and Recovery Reference* for more information about the RMAN `DUPLICATE` and `CONVERT DATABASE` commands
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_TDB` package

Performing a Database Maintenance Operation Using Streams

This section describes performing one of the following database maintenance operations on an Oracle Database 10*g* Release 2 database:

- Migrating the database to a different platform
- Migrating the database to a different character set
- Modifying database schema objects to support upgrades to user-created applications
- Applying an Oracle Database software patch

You can use Streams to achieve little or no downtime during these operations. During the operation, the source database is the existing database on which you are performing database maintenance. The capture database is the database on which the Streams **capture process** runs. The **destination database** is the database that will replace the source database at the end of the operation.

Complete the following tasks to perform a database maintenance operation using Streams:

- Task 1: Beginning the Maintenance Operation
- Task 2: Setting Up Streams Prior to Instantiation
- Task 3: Instantiating the Database
- Task 4: Setting Up Streams After Instantiation
- Task 5: Finishing the Maintenance Operation and Removing Streams

Task 1: Beginning the Maintenance Operation

Complete the following steps to begin the maintenance operation using Oracle Streams:

1. Create an empty Oracle Database 10*g* Release 2 database. This database will be the **destination database** during the maintenance operation. If you are migrating the database to a different platform, then create the database on a computer system that uses the new platform. If you are migrating the database to a different character set, then create a database that uses the new character set.

   See the Oracle installation guide for your operating system if you need to install Oracle, and see the *Oracle Database Administrator's Guide* for information about creating a database.

Make sure the destination database has a different global name than the source database. This example assumes that the global name of the source database is `orcl.net` and the global name of the destination database during the database maintenance operation is `stms.net`. The global name of the destination database is changed when the destination database replaces the source database at the end of the maintenance operation.

2. Make sure the source database is running in `ARCHIVELOG` mode. See *Oracle Database Administrator's Guide* for information about running a database in `ARCHIVELOG` mode.
3. Make sure the initialization parameters are set properly at both databases to support a Streams environment. See "Setting Initialization Parameters Relevant to Streams" on page 10-4.
4. Configure a Streams administrator at each database, including the source database, destination database, and capture database (if the capture database is a third database). See "Configuring a Streams Administrator" on page 10-1 for instructions. This example assumes that the name of the Streams administrator is `strmadmin` at each database.
5. If you are upgrading user-created applications, then supplementally log any columns at the source database that will be involved in a **rule-based transformation**, **DML handler**, or **value dependency**. These columns must be unconditionally logged at the source database. See *Oracle Streams Replication Administrator's Guide* for information about specifying unconditional **supplemental log group**s for these columns.
6. At the source database, make read-only any database objects that are not supported by Streams. To list unsupported database objects, query the `DBA_STREAMS_UNSUPPORTED` data dictionary view.

   "Preparing for Maintenance of a Database with User-defined Types" on page C-7 discusses a method for retaining changes to tables that contain user-defined types during the maintenance operation. If you are using this method, then tables that contain user-defined types can remain open during the maintenance operation.

Task 2: Setting Up Streams Prior to Instantiation

The specific instructions for setting up Streams prior to **instantiation** depend on which database is the capture database. The `PRE_INSTANTIATION_SETUP` procedure always configures the capture process on the database where it is run. Therefore, this procedure must be run at the capture database.

When you run this procedure, you can specify that the procedure performs the configuration directly, or that the procedure generates a script that contains the configuration actions. See "Deciding Whether to Configure Streams Directly or Generate a Script" on page C-10. The examples in this section specify that the procedure performs the configuration directly.

Follow the instructions in the appropriate section:

- The Source Database Is the Capture Database
- The Destination Database Is the Capture Database
- A Third Database Is the Capture Database

Note: When the PRE_INSTANTIATION_SETUP procedure is running with the perform_actions parameter set to true, metadata about its configuration actions is recorded in the following data dictionary views: DBA_RECOVERABLE_SCRIPT, DBA_RECOVERABLE_SCRIPT_PARAMS, DBA_RECOVERABLE_SCRIPT_BLOCKS, and DBA_RECOVERABLE_SCRIPT_ERRORS. If the procedure stops because it encounters an error, then you can use the RECOVER_OPERATION procedure in the DBMS_STREAMS_ADM package to complete the configuration after you correct the conditions that caused the error. These views are not populated if a script is used to configure the **replication** environment.

See Also:

- "Overview of Using Streams for Database Maintenance Operations" on page C-2 for information about the capture database
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the RECOVER_OPERATION procedure

The Source Database Is the Capture Database

Complete the following steps to set up Streams prior to instantiation when the source database is the capture database:

1. Configure your network and Oracle Net so that the source database can communicate with the **destination database**. See *Oracle Database Net Services Administrator's Guide* for instructions.
2. Connect as the Streams administrator in SQL*Plus to the source database, and create a database link to the destination database. For example:

```
CONNECT strmadmin/strmadminpw@orcl.net

CREATE DATABASE LINK stms.net CONNECT TO strmadmin IDENTIFIED BY strmadminpw
   USING 'stms.net';
```

3. While connected as the Streams administrator in SQL*Plus to the source database, run the PRE_INSTANTIATION_SETUP procedure:

```
DECLARE
  empty_tbs  DBMS_STREAMS_TABLESPACE_ADM.TABLESPACE_SET;
BEGIN
  DBMS_STREAMS_ADM.PRE_INSTANTIATION_SETUP(
    maintain_mode           => 'GLOBAL',
    tablespace_names        => empty_tbs,
    source_database         => 'orcl.net',
    destination_database    => 'stms.net',
    perform_actions         => true,
    script_name             => NULL,
    script_directory_object => NULL,
    capture_name            => 'capture_maint',
    capture_queue_table     => 'strmadmin.capture_q_table',
    capture_queue_name      => 'strmadmin.capture_q',
    propagation_name        => 'prop_maint',
    apply_name              => 'apply_maint',
    apply_queue_table       => 'strmadmin.apply_q',
    apply_queue_name        => 'strmadmin.apply_q_table',
```

```
    bi_directional          => false,
    include_ddl             => true,
    start_processes         => false,
    exclude_schemas         => '*',
    exclude_flags           => DBMS_STREAMS_ADM.EXCLUDE_FLAGS_UNSUPPORTED +
                               DBMS_STREAMS_ADM.EXCLUDE_FLAGS_DML +
                               DBMS_STREAMS_ADM.EXCLUDE_FLAGS_DDL);
END;
/
```

4. Proceed to "Task 3: Instantiating the Database" on page C-17.

The Destination Database Is the Capture Database

Complete the following steps to set up Streams prior to instantiation when the **destination database** is the capture database:

1. Configure your network and Oracle Net so that the source database and destination database can communicate with each other. See *Oracle Database Net Services Administrator's Guide* for instructions.
2. Make sure log file shipping from the source database to the destination database is configured. See "Preparing for Downstream Capture" on page C-6 for instructions.
3. Connect as the Streams administrator in SQL*Plus to the destination database, and create a database link to the source database. For example:

```
CONNECT strmadmin/strmadminpw@stms.net

CREATE DATABASE LINK orcl.net CONNECT TO strmadmin IDENTIFIED BY strmadminpw
   USING 'orcl.net';
```

4. While connected as the Streams administrator in SQL*Plus to the destination database, run the `PRE_INSTANTIATION_SETUP` procedure:

```
DECLARE
  empty_tbs  DBMS_STREAMS_TABLESPACE_ADM.TABLESPACE_SET;
BEGIN
  DBMS_STREAMS_ADM.PRE_INSTANTIATION_SETUP(
    maintain_mode           => 'GLOBAL',
    tablespace_names        => empty_tbs,
    source_database         => 'orcl.net',
    destination_database    => 'stms.net',
    perform_actions         => true,
    script_name             => NULL,
    script_directory_object => NULL,
    capture_name            => 'capture_maint',
    capture_queue_table     => 'strmadmin.streams_q_table',
    capture_queue_name      => 'strmadmin.streams_q',
    apply_name              => 'apply_maint',
    apply_queue_table       => 'strmadmin.streams_q',
    apply_queue_name        => 'strmadmin.streams_q_table',
    bi_directional          => false,
    include_ddl             => true,
    start_processes         => false,
    exclude_schemas         => '*',
    exclude_flags           => DBMS_STREAMS_ADM.EXCLUDE_FLAGS_UNSUPPORTED +
                               DBMS_STREAMS_ADM.EXCLUDE_FLAGS_DML +
                               DBMS_STREAMS_ADM.EXCLUDE_FLAGS_DDL);
END;
/
```

Notice that the `propagation_name` parameter is omitted because a **propagation** is not necessary when the destination database is the capture database and the downstream **capture process** and **apply process** use the same **queue** at the destination database.

Also, notice that the capture process and apply process will share a queue named `streams_q` at the destination database.

5. Proceed to "Task 3: Instantiating the Database" on page C-17.

A Third Database Is the Capture Database

This example assumes that the global name of the third database is `thrd.net`. Complete the following steps to set up Streams prior to instantiation when a third database is the capture database:

1. Configure your network and Oracle Net so that the source database, **destination database**, and third database can communicate with each other. See *Oracle Database Net Services Administrator's Guide* for instructions.
2. Make sure log file shipping from the source database to the third database is configured. See "Preparing for Downstream Capture" on page C-6 for instructions.
3. Connect as the Streams administrator in SQL*Plus to the third database, and create a database link to the source database. For example:

```
CONNECT strmadmin/strmadminpw@thrd.net

CREATE DATABASE LINK orcl.net CONNECT TO strmadmin IDENTIFIED BY strmadminpw
   USING 'orcl.net';
```

4. While connected as the Streams administrator in SQL*Plus to the third database, create a database link to the destination database. For example:

```
CREATE DATABASE LINK stms.net CONNECT TO strmadmin IDENTIFIED BY strmadminpw
   USING 'stms.net';
```

5. While connected as the Streams administrator in SQL*Plus to the third database, run the `PRE_INSTANTIATION_SETUP` procedure:

```
DECLARE
  empty_tbs  DBMS_STREAMS_TABLESPACE_ADM.TABLESPACE_SET;
BEGIN
  DBMS_STREAMS_ADM.PRE_INSTANTIATION_SETUP(
    maintain_mode           => 'GLOBAL',
    tablespace_names        => empty_tbs,
    source_database         => 'orcl.net',
    destination_database    => 'stms.net',
    perform_actions         => true,
    script_name             => NULL,
    script_directory_object => NULL,
    capture_name            => 'capture_maint',
    capture_queue_table     => 'strmadmin.capture_q_table',
    capture_queue_name      => 'strmadmin.capture_q',
    propagation_name        => 'prop_maint',
    apply_name              => 'apply_maint',
    apply_queue_table       => 'strmadmin.apply_q',
    apply_queue_name        => 'strmadmin.apply_q_table',
    bi_directional          => false,
    include_ddl             => true,
```

```
    start_processes         => false,
    exclude_schemas         => '*',
    exclude_flags           => DBMS_STREAMS_ADM.EXCLUDE_FLAGS_UNSUPPORTED +
                               DBMS_STREAMS_ADM.EXCLUDE_FLAGS_DML +
                               DBMS_STREAMS_ADM.EXCLUDE_FLAGS_DDL);
END;
/
```

6. Proceed to "Task 3: Instantiating the Database" on page C-17.

Task 3: Instantiating the Database

"Deciding Which Utility to Use for Instantiation" on page C-11 discusses different options for instantiating an entire database. Complete the steps in the appropriate section based on the **instantiation** option you are using:

- Instantiating the Database Using Export/Import
- Instantiating the Database Using the RMAN DUPLICATE Command
- Instantiating the Database Using the RMAN CONVERT DATABASE Command

> **See Also:** *Oracle Streams Replication Administrator's Guide* for more information about performing instantiations

Instantiating the Database Using Export/Import

If you use Oracle Data Pump or original Export/Import to instantiate the **destination database**, then make sure the following parameters are set to the appropriate values:

- Set the `STREAMS_CONFIGURATION` import parameter to `n`.
- If you use original Export/Import, then set the `CONSISTENT` export parameter to `y`. This parameter does not apply to Data Pump exports.
- If you use original Export/Import, then set the `STREAMS_INSTANTIATION` import parameter to `y`. This parameter does not apply to Data Pump imports.

Complete the following steps to instantiate an entire database with Data Pump:

1. Connect in SQL*Plus to the source database as the Streams administrator, and create a directory object to hold the export dump file and export log file:

```
CREATE DIRECTORY dpump_dir AS '/usr/dpump_dir';
```

2. While connected to the source database as the Streams administrator, determine the current system change number (SCN) of the source database:

```
SET SERVEROUTPUT ON SIZE 1000000
DECLARE
  current_scn NUMBER;
BEGIN
  current_scn:= DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER;
      DBMS_OUTPUT.PUT_LINE('Current SCN: ' || current_scn);
END;
/
```

   The returned SCN value is specified for the `FLASHBACK_SCN` Data Pump export parameter in Step 3. Specifying the `FLASHBACK_SCN` export parameter, or a similar export parameter, ensures that the export is consistent to a single SCN. In this example, assume that the query returned `876606`.

After you perform this query, make sure no DDL changes are made to the objects being exported until after the export is complete.

3. On a command line, use Data Pump to export the source database.

   Perform the export by connecting as an administrative user who is granted `EXP_FULL_DATABASE` role. This user also must have `READ` and `WRITE` privilege on the directory object created in Step 1. This example connects as the Streams administrator `strmadmin`.

   The following example is a Data Pump export command:

   ```
   expdp strmadmin/strmadminpw FULL DIRECTORY=DPUMP_DIR DUMPFILE=orc1.dmp
   FLASHBACK_SCN=876606
   ```

   See Also: *Oracle Database Utilities* for information about performing a Data Pump export

4. Connect in SQL*Plus to the destination database as the Streams administrator, and create a directory object to hold the import dump file and import log file:

   ```
   CREATE DIRECTORY dpump_dir AS '/usr/dpump_dir';
   ```

5. Transfer the Data Pump export dump file `orc1.dmp` to the destination database. You can use the `DBMS_FILE_TRANSFER` package, binary FTP, or some other method to transfer the file to the destination database. After the file transfer, the export dump file should reside in the directory that corresponds to the directory object created in Step 4.

6. On a command line at the destination database, use Data Pump to import the export dump file `orc1.dmp`. Make sure no changes are made to the database tables until the import is complete. Performing the import automatically sets the **instantiation SCN** for the destination database and all of its objects.

   Perform the import by connecting as an administrative user who is granted `IMP_FULL_DATABASE` role. This user also must have `READ` and `WRITE` privilege on the directory object created in Step 4. This example connects as the Streams administrator `strmadmin`.

   The following example is an import command:

   ```
   impdp strmadmin/strmadminpw FULL DIRECTORY=DPUMP_DIR DUMPFILE=orc1.dmp STREAMS_
   CONFIGURATION=n
   ```

   See Also: *Oracle Database Utilities* for information about performing a Data Pump import

Instantiating the Database Using the RMAN DUPLICATE Command

If you use the RMAN `DUPLICATE` command for instantiation on the same platform, then complete the following steps:

1. Create a backup of the source database if one does not exist. RMAN requires a valid backup for duplication. In this example, create a backup of `orc1.net` if one does not exist.

2. While connected as an administrative user in SQL*Plus to the source database, determine the until SCN for the RMAN DUPLICATE command. For example:

```
CONNECT SYSTEM/MANAGER@orcl.net

SET SERVEROUTPUT ON SIZE 1000000
DECLARE
  until_scn NUMBER;
BEGIN
  until_scn:= DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER;
      DBMS_OUTPUT.PUT_LINE('Until SCN: ' || until_scn);
END;
/
```

   Make a note of the until SCN value. This example assumes that the until SCN value is 748044. You will set the UNTIL SCN option to this value when you use RMAN to duplicate the database in Step 5 and as the **instantiation SCN** in "Task 4: Setting Up Streams After Instantiation" on page C-23.

3. While still connected as an administrative user in SQL*Plus to the source database, archive the current online redo log. For example:

```
ALTER SYSTEM ARCHIVE LOG CURRENT;
```

4. Prepare your environment for database duplication, which includes preparing the **destination database** as an auxiliary instance for duplication. See the *Oracle Database Backup and Recovery Advanced User's Guide* for instructions.

5. Use the RMAN DUPLICATE command with the OPEN RESTRICTED option to instantiate the source database at the destination database. The OPEN RESTRICTED option is required. This option enables a restricted session in the duplicate database by issuing the following SQL statement: ALTER SYSTEM ENABLE RESTRICTED SESSION. RMAN issues this statement immediately before the duplicate database is opened.

   You can use the UNTIL SCN clause to specify an SCN for the duplication. Use the until SCN determined in Step 2 for this clause. Archived redo logs must be available for the until SCN specified and for higher SCN values. Therefore, Step 3 archived the redo log containing the until SCN.

   Make sure you use TO *database_name* in the DUPLICATE command to specify the name of the duplicate database. In this example, the duplicate database is stms.net. Therefore, the DUPLICATE command for this example includes TO stms.net.

   The following example is an RMAN DUPLICATE command:

```
rman
RMAN> CONNECT TARGET SYS/change_on_install@orcl.net
RMAN> CONNECT AUXILIARY SYS/change_on_install@stms.net
RMAN> RUN
      {
        SET UNTIL SCN 748044;
        ALLOCATE AUXILIARY CHANNEL mgdb DEVICE TYPE sbt;
        DUPLICATE TARGET DATABASE TO mgdb
        NOFILENAMECHECK
        OPEN RESTRICTED;
      }
```

6. Connect to the destination database as a system administrator in SQL*Plus, and rename the global name. After an RMAN database instantiation, the destination database has the same global name as the source database, but the destination database must have its original name until the end of the maintenance operation. Rename the global name of the destination database back to its original name with the following statement:

```
ALTER DATABASE RENAME GLOBAL_NAME TO stms.net;
```

7. Connect as the Streams administrator in SQL*Plus to the destination database, and create a database link to the source database. For example:

```
CONNECT strmadmin/strmadminpw@stms.net

CREATE DATABASE LINK orcl.net CONNECT TO strmadmin IDENTIFIED BY strmadminpw
   USING 'orcl.net';
```

   This database link is required because the `POST_INSTANTIATION_SETUP` procedure runs the `SET_GLOBAL_INSTANTIATION_SCN` procedure in the `DBMS_APPLY_ADM` package at the destination database, and the `SET_GLOBAL_INSTANTIATION_SCN` procedure requires the database link.

8. If the source database and the capture database are the same database, then while still connected as the Streams administrator in SQL*Plus to the destination database, drop the database link from the source database to the destination database that was cloned from the source database:

```
DROP DATABASE LINK stms.net;
```

Instantiating the Database Using the RMAN CONVERT DATABASE Command

If you use the RMAN `CONVERT DATABASE` command for instantiation to migrate the database to a different platform, then complete the following steps:

1. Create a backup of the source database if one does not exist. RMAN requires a valid backup. In this example, create a backup of `orcl.net` if one does not exist.
2. While still connected as an administrative user in SQL*Plus to the source database, archive the current online redo log. For example:

```
ALTER SYSTEM ARCHIVE LOG CURRENT;
```

3. Prepare your environment for database conversion, which includes opening the source database in read-only mode. Complete the following steps:
   a. If the source database is open, then shut it down and start it in read-only mode.
   b. Run the `CHECK_DB` and `CHECK_EXTERNAL` functions in the `DBMS_TDB` package. Check the results to ensure that the conversion is supported by the RMAN `CONVERT DATABASE` command.

      See Also: *Oracle Database Backup and Recovery Advanced User's Guide* for more information about these steps

4. Determine the current SCN of the source database:

```
SET SERVEROUTPUT ON SIZE 1000000
DECLARE
  current_scn NUMBER;
BEGIN
  current_scn:= DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER;
      DBMS_OUTPUT.PUT_LINE('Current SCN: ' || current_scn);
END;
/
```

   Make a note of the SCN value returned. You will use this number for the **instantiation SCN** in "Task 4: Setting Up Streams After Instantiation" on page C-23. For this example, assume that the returned value is `748044`.

5. Open RMAN and run the `CONVERT DATABASE` command.

   Make sure you use `NEW DATABASE` *`database_name`* in the `CONVERT DATABASE` command to specify the name of the **destination database**. In this example, the destination database name is `stms`. Therefore, the `CONVERT DATABASE` command for this example includes `NEW DATABASE stms`.

   The following example is an RMAN `CONVERT DATABASE` command for a destination database that is running on the `Linux IA (64-bit)` platform:

```
rman
RMAN> CONNECT TARGET SYS/change_on_install@orc1.net
CONVERT DATABASE NEW DATABASE 'stms'
          TRANSPORT SCRIPT '/tmp/convertdb/transportscript.sql'
          TO PLATFORM 'Linux IA (64-bit)'
          DB_FILE_NAME_CONVERT = ('/home/oracle/dbs','/tmp/convertdb');
```

6. Transfer the datafiles, PFILE, and SQL script produced by the RMAN `CONVERT DATABASE` command to the computer system that is running the destination database.

7. On the computer system that is running the destination database, modify the SQL script so that the destination database always opens with restricted session enabled.

   An example script follows with the necessary modifications in bold font:

```
-- The following commands will create a new control file and use it
-- to open the database.
-- Data used by Recovery Manager will be lost.
-- The contents of online logs will be lost and all backups will
-- be invalidated. Use this only if online logs are damaged.

-- After mounting the created controlfile, the following SQL
-- statement will place the database in the appropriate
-- protection mode:
--  ALTER DATABASE SET STANDBY DATABASE TO MAXIMIZE PERFORMANCE

STARTUP NOMOUNT PFILE='init_00gd2lak_1_0.ora'
CREATE CONTROLFILE REUSE SET DATABASE "STMS" RESETLOGS  NOARCHIVELOG
    MAXLOGFILES 32
    MAXLOGMEMBERS 2
    MAXDATAFILES 32
    MAXINSTANCES 1
    MAXLOGHISTORY 226
```

```
LOGFILE
  GROUP 1 '/tmp/convertdb/archlog1'  SIZE 25M,
  GROUP 2 '/tmp/convertdb/archlog2'  SIZE 25M
DATAFILE
  '/tmp/convertdb/systemdf',
  '/tmp/convertdb/sysauxdf',
  '/tmp/convertdb/datafile1',
  '/tmp/convertdb/datafile2',
  '/tmp/convertdb/datafile3'
CHARACTER SET WE8DEC
;

-- NOTE: This ALTER SYSTEM statement is added to enable restricted session.

ALTER SYSTEM ENABLE RESTRICTED SESSION;

-- Database can now be opened zeroing the online logs.
ALTER DATABASE OPEN RESETLOGS;

-- No tempfile entries found to add.
--

set echo off
prompt ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
prompt * Your database has been created successfully!
prompt * There are many things to think about for the new database. Here
prompt * is a checklist to help you stay on track:
prompt * 1. You may want to redefine the location of the directory objects.
prompt * 2. You may want to change the internal database identifier (DBID)
prompt *    or the global database name for this database. Use the
prompt *    NEWDBID Utility (nid).
prompt ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

SHUTDOWN IMMEDIATE
-- NOTE: This startup has the UPGRADE parameter.
-- The startup already has restricted session enabled, so no change is needed.
STARTUP UPGRADE PFILE='init_00gd2lak_1_0.ora'
@@ ?/rdbms/admin/utlirp.sql
SHUTDOWN IMMEDIATE
-- NOTE: The startup below is generated without the RESTRICT clause.
-- Add the RESTRICT clause.
STARTUP RESTRICT PFILE='init_00gd2lak_1_0.ora'
-- The following step will recompile all PL/SQL modules.
-- It may take serveral hours to complete.
@@ ?/rdbms/admin/utlrp.sql
set feedback 6;
```

Other changes to the script might be necessary. For example, the datafile locations and PFILE location might need to be changed to point to the correct locations on the destination database computer system.

8. Connect to the destination database as a system administrator in SQL*Plus, and rename the global name. After an RMAN database instantiation, the destination database has the same global name as the source database, but the destination database must have its original name until the end of the maintenance operation. Rename the global name of the destination database back to its original name with the following statement:

```
ALTER DATABASE RENAME GLOBAL_NAME TO stms.net;
```

9. Connect as the Streams administrator in SQL*Plus to the destination database, and create a database link to the source database. For example:

```
CONNECT strmadmin/strmadminpw@stms.net

CREATE DATABASE LINK orcl.net CONNECT TO strmadmin IDENTIFIED BY strmadminpw
   USING 'orcl.net';
```

   This database link is required because the POST_INSTANTIATION_SETUP procedure runs the SET_GLOBAL_INSTANTIATION_SCN procedure in the DBMS_APPLY_ADM package at the destination database, and the SET_GLOBAL_INSTANTIATION_SCN procedure requires the database link.

10. If the source database and the capture database are the same database, then while still connected as the Streams administrator in SQL*Plus to the destination database, drop the database link from the source database to the destination database that was cloned from the source database:

```
DROP DATABASE LINK stms.net;
```

Task 4: Setting Up Streams After Instantiation

To set up Streams after **instantiation**, run the POST_INSTANTIATION_SETUP procedure. The POST_INSTANTIATION_SETUP procedure must be run at the database where the PRE_INSTANTIATION_SETUP procedure was run in "Task 2: Setting Up Streams Prior to Instantiation" on page C-13.

When you run the POST_INSTANTIATION_SETUP procedure, you can specify that the procedure performs the configuration directly, or that the procedure generates a script that contains the configuration actions. See "Deciding Whether to Configure Streams Directly or Generate a Script" on page C-10. The examples in this section specify that the procedure performs the configuration directly.

The parameter values specified in the PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP procedures must match, except for the values of the following parameters: perform_actions, script_name, script_directory_object, and start_processes. In this example, all of the parameter values match in the two procedures.

It is important to set the instantiation_scn parameter in the POST_INSTANTIATION_SETUP procedure correctly. Follow these instructions when you set this parameter:

- If RMAN was used for instantiation, then set the instantiation_scn parameter to the value determined during instantiation. This value was determined when you completed the instantiation in "Instantiating the Database Using the RMAN DUPLICATE Command" on page C-18 or "Instantiating the Database Using the RMAN CONVERT DATABASE Command" on page C-20.

  The source database and third database examples in this section set the instantiation_scn parameter to 748044 for the following reasons:

  – If the RMAN DUPLICATE command was used for instantiation, then the command duplicates the database up to one less than the SCN value specified in the UNTIL SCN clause. Therefore, you should subtract one from the until SCN value that you specified when you ran the DUPLICATE command in Step 5 on page C-19 in "Instantiating the Database Using the RMAN DUPLICATE Command". In this example, the until SCN was set to 748045. Therefore, the instantiation_scn parameter should be set to 748045 - 1, or 748044.

  - If the RMAN CONVERT DATABASE command was used for instantiation, then the instantiation_scn parameter should be set to the SCN value determined immediately before running the CONVERT DATABASE command. This value was determined in Step 4 on page C-21 in "Instantiating the Database Using the RMAN CONVERT DATABASE Command".

- If Export/Import was used for instantiation, then the **instantiation SCN** was set during import, and the instantiation_scn parameter must be set to NULL. The **destination database** example in this section sets the instantiation_scn to NULL because RMAN cannot be used for database instantiation when the destination database is the capture database.

The specific instructions for setting up Streams after instantiation depend on which database is the capture database. Follow the instructions in the appropriate section:

- The Source Database Is the Capture Database
- The Destination Database Is the Capture Database
- A Third Database Is the Capture Database

Note: When the POST_INSTANTIATION_SETUP procedure is running with the perform_actions parameter set to true, metadata about its configuration actions is recorded in the following data dictionary views: DBA_RECOVERABLE_SCRIPT, DBA_RECOVERABLE_SCRIPT_PARAMS, DBA_RECOVERABLE_SCRIPT_BLOCKS, and DBA_RECOVERABLE_SCRIPT_ERRORS. If the procedure stops because it encounters an error, then you can use the RECOVER_OPERATION procedure in the DBMS_STREAMS_ADM package to complete the configuration after you correct the conditions that caused the error. These views are not populated if a script is used to configure the **replication** environment.

See Also:

- "Overview of Using Streams for Database Maintenance Operations" on page C-2 for information about the capture database
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the RECOVER_OPERATION procedure

The Source Database Is the Capture Database

Complete the following steps to set up Streams after instantiation when the source database is the capture database:

1. Connect in SQL*Plus to the source database as the Streams administrator, and run the POST_INSTANTIATION_SETUP procedure:

```
CONNECT strmadmin/strmadminpw@orcl.net

DECLARE
  empty_tbs  DBMS_STREAMS_TABLESPACE_ADM.TABLESPACE_SET;
BEGIN
  DBMS_STREAMS_ADM.POST_INSTANTIATION_SETUP(
    maintain_mode           => 'GLOBAL',
    tablespace_names        => empty_tbs,
    source_database         => 'orcl.net',
    destination_database    => 'stms.net',
```

```
    perform_actions         => true,
    script_name             => NULL,
    script_directory_object => NULL,
    capture_name            => 'capture_maint',
    capture_queue_table     => 'strmadmin.capture_q_table',
    capture_queue_name      => 'strmadmin.capture_q',
    propagation_name        => 'prop_maint',
    apply_name              => 'apply_maint',
    apply_queue_table       => 'strmadmin.apply_q',
    apply_queue_name        => 'strmadmin.apply_q_table',
    bi_directional          => false,
    include_ddl             => true,
    start_processes         => false,
    instantiation_scn       => 748044, -- NULL if Export/Import instantiation
    exclude_schemas         => '*',
    exclude_flags           => DBMS_STREAMS_ADM.EXCLUDE_FLAGS_UNSUPPORTED +
                               DBMS_STREAMS_ADM.EXCLUDE_FLAGS_DML +
                               DBMS_STREAMS_ADM.EXCLUDE_FLAGS_DDL);
END;
/
```

2. Proceed to "Task 5: Finishing the Maintenance Operation and Removing Streams" on page C-26.

The Destination Database Is the Capture Database

Complete the following steps to set up Streams after instantiation when the **destination database** is the capture database:

1. Connect in SQL*Plus to the destination database as the Streams administrator, and run the `POST_INSTANTIATION_SETUP` procedure:

```
CONNECT strmadmin/strmadminpw@stms.net

DECLARE
  empty_tbs  DBMS_STREAMS_TABLESPACE_ADM.TABLESPACE_SET;
BEGIN
  DBMS_STREAMS_ADM.POST_INSTANTIATION_SETUP(
    maintain_mode           => 'GLOBAL',
    tablespace_names        => empty_tbs,
    source_database         => 'orcl.net',
    destination_database    => 'stms.net',
    perform_actions         => true,
    script_name             => NULL,
    script_directory_object => NULL,
    capture_name            => 'capture_maint',
    capture_queue_table     => 'strmadmin.streams_q_table',
    capture_queue_name      => 'strmadmin.streams_q',
    apply_name              => 'apply_maint',
    apply_queue_table       => 'strmadmin.streams_q',
    apply_queue_name        => 'strmadmin.streams_q_table',
    bi_directional          => false,
    include_ddl             => true,
    start_processes         => false,
    instantiation_scn       => NULL,
    exclude_schemas         => '*',
    exclude_flags           => DBMS_STREAMS_ADM.EXCLUDE_FLAGS_UNSUPPORTED +
                               DBMS_STREAMS_ADM.EXCLUDE_FLAGS_DML +
                               DBMS_STREAMS_ADM.EXCLUDE_FLAGS_DDL);
END;
/
```

Notice that the `propagation_name` parameter is omitted because a **propagation** is not necessary when the destination database is the capture database.

2. Proceed to "Task 5: Finishing the Maintenance Operation and Removing Streams" on page C-26.

A Third Database Is the Capture Database

This example assumes that the global name of the third database is `thrd.net`. Complete the following steps to set up Streams after instantiation when a third database is the capture database:

1. Connect in SQL*Plus to the third database as the Streams administrator, and run the `POST_INSTANTIATION_SETUP` procedure:

```
CONNECT strmadmin/strmadminpw@thrd.net

DECLARE
  empty_tbs  DBMS_STREAMS_TABLESPACE_ADM.TABLESPACE_SET;
BEGIN
  DBMS_STREAMS_ADM.POST_INSTANTIATION_SETUP(
    maintain_mode           => 'GLOBAL',
    tablespace_names        => empty_tbs,
    source_database         => 'orcl.net',
    destination_database    => 'stms.net',
    perform_actions         => true,
    script_name             => NULL,
    script_directory_object => NULL,
    capture_name            => 'capture_maint',
    capture_queue_table     => 'strmadmin.capture_q_table',
    capture_queue_name      => 'strmadmin.capture_q',
    propagation_name        => 'prop_maint',
    apply_name              => 'apply_maint',
    apply_queue_table       => 'strmadmin.apply_q',
    apply_queue_name        => 'strmadmin.apply_q_table',
    bi_directional          => false,
    include_ddl             => true,
    start_processes         => false,
    instantiation_scn       => 748044, -- NULL if Export/Import instantiation
    exclude_schemas         => '*',
    exclude_flags           => DBMS_STREAMS_ADM.EXCLUDE_FLAGS_UNSUPPORTED +
                               DBMS_STREAMS_ADM.EXCLUDE_FLAGS_DML +
                               DBMS_STREAMS_ADM.EXCLUDE_FLAGS_DDL);
END;
/
```

2. Proceed to "Task 5: Finishing the Maintenance Operation and Removing Streams" on page C-26.

Task 5: Finishing the Maintenance Operation and Removing Streams

Complete the following steps to finish a maintenance operation using Oracle Streams and remove Streams components:

1. At the **destination database**, disable any imported jobs that modify data that will be replicated from the source database. Query the `DBA_JOBS` data dictionary view to list the jobs.
2. If you are applying a patch, then apply the patch to the destination database now. Follow the instructions included with the patch.

3. If you are upgrading user-created applications, then, at the destination database, you might need to complete the following steps:

   a. Modify the schema objects in the database to support the upgraded user-created applications.

   b. Configure one or more **declarative rule-based transformation**s and **DML handler**s that modify row LCRs from the source database so that the apply process applies these row LCRs to the modified schema objects correctly. For example, if a column name was changed to support the upgraded user-created applications, then a declarative rule-based transformation should rename the column in a row LCR that involves the column.

      See "Handling Modifications to Schema Objects" on page C-8.

   c. Configure one or more **virtual dependency definition**s if row LCRs might contain logical dependencies that cannot be detected by the apply process alone.

      See "Handling Logical Dependencies" on page C-9.

4. While connected as an administrative user in SQL*Plus to the destination database, use the `ALTER SYSTEM` statement to disable the `RESTRICTED SESSION`:

```
CONNECT SYSTEM/MANAGER

ALTER SYSTEM DISABLE RESTRICTED SESSION;
```

5. While connected as the Streams administrator in SQL*Plus to the destination database, start the **apply process**. For example:

```
CONNECT strmadmin/strmadminpw@stms.net

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

6. While connected as the Streams administrator in SQL*Plus to the capture database, start the capture process. For example:

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

   This step begins the process of replicating changes that were made to the source database during **instantiation** of the destination database.

7. Monitor the Streams environment until the apply process at the destination database has applied most of the changes from the source database. For example, if the name of the capture process is `capture_maint`, and the name of the apply process is `apply_maint`, then run the following query at the capture database:

```
COLUMN ENQUEUE_MESSAGE_NUMBER HEADING 'Captured SCN' FORMAT 99999999999
COLUMN LWM_MESSAGE_NUMBER HEADING 'Applied SCN' FORMAT 99999999999

SELECT c.ENQUEUE_MESSAGE_NUMBER, a.LWM_MESSAGE_NUMBER
  FROM V$STREAMS_CAPTURE c, V$STREAMS_APPLY_COORDINATOR@stms.net a
  WHERE CAPTURE_NAME = 'CAPTURE_MAINT'
    AND APPLY_NAME   = 'APPLY_MAINT';
```

When the two SCN values returned by this query are nearly equal, most of the changes from the source database have been applied at the destination database, and you can move on to the next step. At this point in the process, the values returned by this query might never be equal because the source database still allows changes.

If this query returns no results, then make sure the **Streams client**s in the environment are enabled by querying the `STATUS` column in the `DBA_CAPTURE` view at the capture database and the `DBA_APPLY` view at the destination database. If the Streams configuration uses a **propagation**, you can check the status of the propagation by running the query in "Displaying the Schedule for a Propagation Job" on page 21-16.

If a Streams client is disabled, then try restarting it. If a Streams client will not restart, then troubleshoot the environment using the information in Chapter 18, "Troubleshooting a Streams Environment".

8. While connected as the Streams administrator in SQL*Plus to the destination database, make sure there are no apply errors by running the following query:

```
CONNECT strmadmin/strmadminpw@stms.net

SELECT COUNT(*) FROM DBA_APPLY_ERROR;
```

If this query returns zero, then move on to the next step. If this query shows errors in the error queue, then resolve these errors before continuing. See "Managing Apply Errors" on page 13-23 for instructions.

9. Disconnect all applications and users from the source database.

10. While connected as an administrative user in SQL*Plus to the source database, restrict access to the database. For example:

```
CONNECT SYSTEM/MANAGER@orcl.net

ALTER SYSTEM ENABLE RESTRICTED SESSION;
```

11. While connected as an administrative user in SQL*Plus to the source database, repeat the query you ran in Step 7. When the two SCN values returned by the query are equal, all of the changes from the source database have been applied at the destination database, and you can move on to the next step.

12. While connected as the Streams administrator in SQL*Plus to the destination database, repeat the query you ran in Step 8. If this query returns zero, then move on to the next step. If this query shows errors in the error queue, then resolve these errors before continuing. See "Managing Apply Errors" on page 13-23 for instructions.

13. If you performed any actions that created, modified, or deleted job queue processes at the source database during the maintenance operation, then perform the same actions at the destination database. See "Considerations for Job Queue Processes and PL/SQL Package Subprograms" on page C-4 for more information.

14. If you invoked any Oracle-supplied PL/SQL package subprograms at the source database during the maintenance operation that modified both user data and dictionary metadata at the same time, then invoke the same subprograms at the destination database. See "Considerations for Job Queue Processes and PL/SQL Package Subprograms" on page C-4 for more information.

15. Remove the Streams components that are no longer needed from both databases, including the ANYDATA queues, **supplemental logging** specifications, the capture process, the propagation if one exists, and the apply process. Connect as the Streams administrator in SQL*Plus to the capture database, and run the CLEANUP_INSTANTIATION_SETUP procedure to remove the Streams components both databases.

    If the capture database is the source database or a third database, then run the following procedure:

```
DECLARE
  empty_tbs  DBMS_STREAMS_TABLESPACE_ADM.TABLESPACE_SET;
BEGIN
  DBMS_STREAMS_ADM.CLEANUP_INSTANTIATION_SETUP(
    maintain_mode           => 'GLOBAL',
    tablespace_names        => empty_tbs,
    source_database         => 'orcl.net',
    destination_database    => 'stms.net',
    perform_actions         => true,
    script_name             => NULL,
    script_directory_object => NULL,
    capture_name            => 'capture_maint',
    capture_queue_table     => 'strmadmin.capture_q_table',
    capture_queue_name      => 'strmadmin.capture_q',
    propagation_name        => 'prop_maint',
    apply_name              => 'apply_maint',
    apply_queue_table       => 'strmadmin.apply_q',
    apply_queue_name        => 'strmadmin.apply_q_table',
    bi_directional          => false,
    change_global_name      => true);
END;
/
```

    If the capture database is the destination database, then run the following procedure:

```
DECLARE
  empty_tbs  DBMS_STREAMS_TABLESPACE_ADM.TABLESPACE_SET;
BEGIN
  DBMS_STREAMS_ADM.CLEANUP_INSTANTIATION_SETUP(
    maintain_mode           => 'GLOBAL',
    tablespace_names        => empty_tbs,
    source_database         => 'orcl.net',
    destination_database    => 'stms.net',
    perform_actions         => true,
    script_name             => NULL,
    script_directory_object => NULL,
    capture_name            => 'capture_maint',
    capture_queue_table     => 'strmadmin.streams_q_table',
    capture_queue_name      => 'strmadmin.streams_q',
    apply_name              => 'apply_maint',
    apply_queue_table       => 'strmadmin.streams_q',
    apply_queue_name        => 'strmadmin.streams_q_table',
    bi_directional          => false,
    change_global_name      => true);
END;
/
```

    Notice that the propagation_name parameter is omitted because a propagation is not necessary when the destination database is the capture database.

Both sample procedures in this step rename the global name of the destination database to `orc1.net` because the `change_global_name` parameter is set to `true`.

16. Shut down the source database. This database should not be opened again.
17. At the destination database, enable any jobs that you disabled earlier.
18. Make the destination database available for applications and users. Redirect any applications and users that were connecting to the source database to the destination database. If necessary, reconfigure your network and Oracle Net so that systems that communicated with the source database now communicate with the destination database. See *Oracle Database Net Services Administrator's Guide* for instructions.
19. If you no longer need the Streams administrator at the destination database, then connect as an administrative user in SQL*Plus to the destination database, and run the following statement:

```
CONNECT SYSTEM/MANAGER@orcl.net

DROP USER strmadmin CASCADE;
```

The maintenance operation is complete.

Glossary

action context

Optional information associated with a **rule** that is interpreted by the client of the **rules engine** when the rule is evaluated for a **message**.

ANYDATA queue

A **queue** of type `ANYDATA`. These queues can stage **message**s of different types wrapped in an `ANYDATA` wrapper.

See Also: **typed queue**

applied SCN

A system change number (SCN) relating to a **capture process** that corresponds to the most recent **message** dequeued by an **apply process** that applies changes captured by the capture process.

apply forwarding

A **directed network** in which **message**s being forwarded at an intermediate database are first processed by an **apply process**. These messages are then recaptured by a **capture process** at the intermediate database and forwarded.

See Also: **queue forwarding**

apply handler

A user-defined procedure used by an **apply process** for customized processing of **message**s. Apply handlers include **message handler**s, **DML handler**s, **DDL handler**s, **precommit handler**s, and **error handler**s.

apply process

An optional **Streams client** that is an Oracle background process that dequeues logical change records (LCRs) and **user message**s from a specific **queue** and either applies each one directly or passes it as a parameter to an **apply handler**.

See Also: **logical change record (LCR)**

apply servers

A component of an **apply process** that includes one or more parallel execution servers which apply LCRs to database objects as DML or DDL statements or pass the LCRs to their appropriate **apply handler**s. For **user message**s, the apply servers pass the **message**s to the **message handler**. Apply servers can also enqueue **logical change record (LCR)** and non-LCR messages into a queue specified by the `DBMS_APPLY_ADM.SET_ENQUEUE_DESTINATION` procedure. If an apply server encounters an error, then it tries to resolve the error with a user-specified **error handler**. If an apply

server cannot resolve an error, then it places the entire transaction, including all of its LCRs, in the error queue.

See Also: **logical change record (LCR)**

apply user

The user in whose security domain an **apply process** dequeues **message**s that satisfy its **rule set**s, applies messages directly to database objects, runs custom **rule-based transformation**s configured for apply process **rule**s, and runs **apply handler**s configured for the apply process.

approximate commit system change number (approximate CSCN)

An SCN value based on the current SCN of the database when a transaction that has enqueued **message**s into a **commit-time queue** is committed.

archived-log downstream capture process

A **downstream capture process** that captures changes in archived redo log files copied from the **source database** to the **downstream database**.

barrier transaction

A DDL transaction or a transaction that includes a **row logical change record (row LCR)** for which an **apply process** cannot identify the table rows or the database object by using the **destination database** data dictionary and **virtual dependency definition**s.

builder server

A component of a **capture process** that is a parallel execution server which merges redo records from the **preparer server**. These redo records either evaluated to `TRUE` during partial evaluation or partial evaluation was inconclusive for them. The builder server preserves the system change number (SCN) order of these redo records and passes the merged redo records to the capture process.

buffered queue

The portion of a **queue** that uses the **Streams pool** to store **message**s in memory and a queue table to store messages that have spilled from memory.

capture process

An optional **Streams client** that is an Oracle background process that scans a database redo log to capture DML and DDL changes made to database objects.

capture user

The user in whose security domain a **capture process** captures changes that satisfy its **rule set**s and runs custom **rule-based transformation**s configured for capture process **rule**s.

captured message

A **message** that was captured implicitly by a **capture process**. A captured message contains a **logical change record (LCR)**.

See Also: **user-enqueued message**

captured SCN

The system change number (SCN) that corresponds to the most recent change scanned in the redo log by a **capture process**.

change cycling

Sending a change back to the database where it originated. Typically, change cycling should be avoided in an information sharing environment by using **tag**s and by using the LCR member function `GET_SOURCE_DATABASE_NAME` in **rule condition**s.

See Also: **logical change record (LCR)**

checkpoint

Information about the current state of a **capture process** that is stored persistently in the data dictionary of the database running the capture process.

checkpoint interval

A regular interval at which a **capture process** attempts to record a **checkpoint**.

checkpoint retention time

The amount of time that a **capture process** retains **checkpoint**s before purging them automatically.

column list

A list of columns for which an update conflict handler is called when an update **conflict** occurs for one or more of the columns in the list.

See Also: **conflict resolution**

commit-time queue

A **queue** in which **message**s are ordered by their **approximate commit system change number (approximate CSCN)** values.

conditional log group

A **supplemental log group** that logs the before images of all specified columns only if at least one of the columns in the supplemental log group is modified.

See Also: **unconditional log group**

conflict

A mismatch between the old values in an LCR and the expected data in a table. Conflicts are detected by an **apply process** when it attempts to apply an LCR. Conflicts typically result when two different databases that are sharing data in a table modify the same row in the table at nearly the same time.

See Also: **logical change record (LCR)**

conflict resolution

Handling a **conflict** to avoid an apply error. Either prebuilt update conflict handlers or custom conflict handlers can resolve conflicts.

consumption

The process of dequeuing an **message** from a **queue**.

coordinator process

A component of an **apply process** that is an Oracle background process that gets transactions from the **reader server** and passes them to **apply servers**.

custom apply

An **apply process** passes an LCR as a parameter to a user procedure for processing. The user procedure can process the LCR in a customized way.

See Also: **logical change record (LCR)**

custom rule-based transformation

A **rule-based transformation** that requires a user-defined PL/SQL function to perform the transformation.

See Also: **declarative rule-based transformation**

database supplemental logging

The type of **supplemental logging** that can apply to the primary key, foreign key, and unique key columns in an entire database.

DDL handler

An **apply handler** that processes DDL LCRs.

See Also: **DDL logical change record (DDL LCR)**

DDL logical change record (DDL LCR)

A **logical change record (LCR)** that describes a data definition language (DDL) change.

declarative rule-based transformation

A **rule-based transformation** that covers one of a common set of transformation scenarios for row LCRs. Declarative rule-based transformations are run internally without using PL/SQL.

See Also: **row logical change record (row LCR)** and **custom rule-based transformation**

direct apply

An **apply process** applies an LCR without running a user procedure.

See Also: **logical change record (LCR)**

directed network

A network in which propagated **message**s pass through one or more intermediate databases before arriving at a **destination database**.

destination database

A database where **message**s are consumed. Messages can be consumed when they are dequeued implicitly from a **queue** by a **propagation** or **apply process**, or messages can be consumed when they are dequeued explicitly from a queue by an application, a **messaging client**, or a user.

See Also: **consumption**

destination queue

The **queue** that receives the **message**s propagated by a **propagation** from a **source queue**.

DML handler

An **apply handler** that processes row LCRs.

See Also: **row logical change record (row LCR)**

downstream capture process

A **capture process** that runs on a database other than its **source database**.

downstream database

The database on which a **downstream capture process** runs.

error handler

An **apply handler** that tries to resolve apply errors. An error handler is invoked only when a **row logical change record (row LCR)** raises an **apply process** error. Such an error might result from a **conflict** if no conflict handler is specified or if the update conflict handler cannot resolve the conflict.

evaluation context

A database object that defines external data that can be referenced in **rule condition**s. The external data can exist as variables, table data, or both.

exception queue

Messages are transferred to an exception **queue** if they cannot be retrieved and processed for some reason.

explicit capture

The **message**s are enqueued into a **queue** by an application or a user.

expression

A combination of one or more values and operators that evaluate to a value.

file

In the context of a **file group**, a reference to a file stored on hard disk. A file is composed of a file name, a directory object, and a file type. The directory object references the directory in which the file is stored on hard disk.

file group

A collection of **version**s.

file group repository

A collection of all of the **file group**s in a database.

first SCN

The lowest system change number (SCN) in the redo log from which a **capture process** can capture changes.

global rule

A **rule** that is relevant either to an entire database or an entire **queue**.

heterogeneous information sharing

Sharing information between Oracle and non-Oracle databases.

high-watermark

The system change number (SCN) beyond which no **message**s have been applied by an **apply process**.

See Also: **low-watermark**

ignore SCN

The system change number (SCN) specified for a table below which changes cannot be applied by an **apply process**.

implicit capture

The **message**s are captured by a **capture process** and enqueued into a **queue**.

instantiation

The process of preparing database objects for instantiation at a **source database**, optionally copying the database objects from a source database to a **destination database**, and setting the **instantiation SCN** for each instantiated database object.

instantiation SCN

The system change number (SCN) for a table which specifies that only changes that were committed after the SCN at the **source database** are applied by an **apply process**.

LCR

See **logical change record (LCR)**.

LOB assembly

An option for DML handlers and error handlers that assembles multiple row LCRs resulting from a change to a single row with LOB columns into a single row LCR. LOB assembly simplifies processing of row LCRs with LOB columns in DML handlers and error handlers.

local capture process

A **capture process** that runs on its **source database**.

logical change record (LCR)

A **message** with a specific format that describes a database change.

See Also: **row logical change record (row LCR)** and **DDL logical change record (DDL LCR)**

LogMiner data dictionary

A separate data dictionary used by a **capture process** to determine the details of a change that it is capturing. The LogMiner data dictionary is necessary because the primary data dictionary of the **source database** might not be synchronized with the redo data being scanned by a capture process.

low-watermark

The system change number (SCN) up to which all **message**s have been applied by an **apply process**.

See Also: **high-watermark**

maximum checkpoint SCN

The system change number (SCN) that corresponds to the last **checkpoint interval** recorded by a **capture process**.

message

A unit of shared information in a Streams environment.

message handler

An **apply handler** that processes **user-enqueued message**s that do not contain LCRs.

See Also: **logical change record (LCR)**

message rule

A **rule** that is relevant only for a **user-enqueued message** of a specific message type.

messaging client

An optional **Streams client** that dequeues **user-enqueued message**s when it is invoked by an application or a user.

negative rule set

A **rule set** for a **Streams client** that results in the Streams client discarding a **message** when a **rule** in the rule set evaluates to `TRUE` for the message. The negative rule set for a Streams client always is evaluated before the **positive rule set**.

nonpersistent

Nonpersistent **queue**s store messages in memory. They are generally used to provide an asynchronous mechanism to send notifications to all users that are currently connected. Nonpersistent queues are deprecated in Oracle Database 10*g* Release 2. Oracle recommends that you use buffered messaging instead.

nontransactional queue

A **queue** in which each **user-enqueued message** is its own transaction.

See Also: **transactional queue**

object dependency

A **virtual dependency definition** that defines a parent-child relationship between two objects at a **destination database**.

oldest SCN

For a running **apply process**, the earliest system change number (SCN) of the transactions currently being dequeued and applied. For a stopped apply process, the oldest SCN is the earliest SCN of the transactions that were being applied when the apply process was stopped.

positive rule set

A **rule set** for a **Streams client** that results in the Streams client performing its task for a **message** when a **rule** in the rule set evaluates to `TRUE` for the message. The **negative rule set** for a Streams client always is evaluated before the positive rule set.

precommit handler

An **apply handler** that can receive the commit information for a transaction and process the commit information in any customized way.

prepared table

A table that has been prepared for **instantiation**.

preparer server

A component of a **capture process** that scans a region defined by the **reader server** and performs prefiltering of changes found in the redo log. A reader server is parallel execution server, and multiple reader servers can run in parallel. Prefiltering entails sending partial information about changes, such as schema and object name for a change, to the **rules engine** for evaluation, and receiving the results of the evaluation.

propagation

An optional **Streams client** that uses a job to send **message**s from a **source queue** to a **destination queue**.

propagation job

A job used by a **propagation** to propagate **message**s.

propagation schedule

A schedule that specifies how often a **propagation job** propagates **message**s.

queue

The abstract storage unit used by a messaging system to store messages.

queue forwarding

A **directed network** in which the **message**s being forwarded at an intermediate database are the messages received by the intermediate database, so that the **source database** for a message is the database where the message originated.

See Also: **apply forwarding**

queue table

A database table where **queue**s are stored. Each queue table contains a default exception queue.

reader server

1. A component of a **capture process** that is a parallel execution server which reads the redo log and divides the redo log into regions.
2. A component of an **apply process** that dequeues **message**s. The reader server is a parallel execution server that computes dependencies between LCRs and assembles messages into transactions. The reader server then returns the assembled transactions to the **coordinator process**, which assigns them to idle **apply servers**.

See Also: **logical change record (LCR)**

real-time downstream capture process

A **downstream capture process** that can capture changes made at the **source database** before the changes are archived in an archived redo log file.

required checkpoint SCN

The system change number (SCN) that corresponds to the lowest **checkpoint interval** for which a **capture process** requires redo data.

replication

The process of sharing database objects and data at multiple databases.

resolution column

The column used to identify a prebuilt update conflict handler.

See Also: **conflict resolution**

row logical change record (row LCR)

A **logical change record (LCR)** that describes a change to the data in a single row or a change to a single `LONG`, `LONG RAW`, or LOB column in a row that results from a data manipulation language (DML) statement or a piecewise update to a LOB. One DML statement can result in multiple row LCRs.

row migration

An automatic conversion performed by an internal **rule-based transformation** when a **subset rule** evaluates to `TRUE` in which an `UPDATE` operation might be converted into an `INSERT` or `DELETE` operation.

rule

A database object that enables a client to perform an action when an event occurs and a condition is satisfied.

rule condition

A component of a **rule** which combines one or more **expression**s and conditions and returns a Boolean value, which is a value of `TRUE`, `FALSE`, or `NULL` (unknown).

rule set

A group of **rule**s.

rule-based transformation

Any modification to a **message** when a **rule** in a **positive rule set** evaluates to `TRUE`.

rules engine

A built-in part of Oracle that evaluates **rule set**s.

schema rule

A **rule** that is relevant only to a particular schema.

secure queue

A **queue** for which AQ agents must be associated explicitly with one or more database users who can perform queue operations, such as enqueue and dequeue.

source database

The database where changes captured by a **capture process** are generated in a redo log.

source queue

The **queue** from which a **propagation** propagates **message**s to a **destination queue**.

start SCN

The system change number (SCN) from which a **capture process** begins to capture changes.

Streams client

A mechanism that performs work in a Streams environment and is a client of the **rules engine** (when the mechanism is associated with one or more **rule set**s). The following are Streams clients: **capture process**, **propagation**, **apply process**, and **messaging client**.

Streams data dictionary

A separate data dictionary used by **propagation**s and **apply process**es to keep track of the database objects from a particular **source database**.

Streams pool

A portion of memory in the System Global Area (SGA) that is used by Streams. The Streams pool stores **buffered queue message**s in memory, and it provides memory for **capture process**es and **apply process**es.

subset rule

A **rule** that is relevant only to a subset of the rows in a particular table.

supplemental log group

A group of columns in a table that is supplementally logged.

See Also: **supplemental logging**

supplemental logging

Additional column data placed in a redo log whenever an operation is performed. A **capture process** captures this additional information and places it in LCRs, and the additional information might be needed for an **apply process** to apply LCRs properly at a **destination database**.

See Also: **logical change record (LCR)**

system-created rule

A **rule** with a system-generated name that was created using the `DBMS_STREAMS_ADM` package.

table rule

A **rule** that is relevant only to a particular table.

table supplemental logging

The type of **supplemental logging** that applies to columns in a particular table.

tablespace repository

A collection of the tablespace sets in a **file group**.

tag

Data of `RAW` datatype that appears in each redo entry and LCR. Tags can be used to modify the behavior of **Streams client**s and to track LCRs. Tags can also be used to prevent **change cycling**.

See Also: **logical change record (LCR)**

transactional queue

A **queue** in which **user-enqueued message**s can be grouped into a set that are applied as one transaction.

See Also: **nontransactional queue**

typed queue

A **queue** that can stage **message**s of one specific type only.

See Also: **ANYDATA queue**

unconditional log group

A **supplemental log group** that logs the before images of specified columns when the table is changed, regardless of whether the change affected any of the specified columns.

See Also: **conditional log group**

user message

A non-LCR **message** of a user-defined type.

See Also: **logical change record (LCR)**

user-enqueued message

A **message** that was enqueued explicitly by an application, a user, or an **apply process**. A user-enqueued message can contain a **logical change record (LCR)** or a **user message**.

See Also: **captured message**

value dependency

A **virtual dependency definition** that defines a table constraint, such as a unique key, or a relationship between the columns of two or more tables.

version

A collection of related **file**s.

virtual dependency definition

A description of a dependency that is used by an **apply process** to detect dependencies between transactions being applied at a **destination database**.

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

X