Oracle® Streams

Concepts and Administration

10*g* Release 1 (10.1)

Part No. B10727-01

December 2003

ORACLE®

Oracle Streams Concepts and Administration, 10*g* Release 1 (10.1)

Part No. B10727-01

Primary Author: Randy Urbano

Graphic Artist: Valarie Moore

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, Jairaj Galagali, Brajesh Goyal, Connie Green, Sanjay Kaluskar, Lewis Kaplan, Joydip Kundu, Anand Lakshminath, Jing Liu, Edwina Lu, Raghu Mani, Pat McElroy, Krishnan Meiyyappan, Shailendra Mishra, Tony Morales, 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

14 Monitoring a Streams Environment

Send Us Your Comments

Oracle Streams Concepts and Administration, 10*g* Release 1 (10.1)

Part No. B10727-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this document. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most?

If you find any errors or have any other suggestions for improvement, please indicate the document title and part number, and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: infodev_us@oracle.com
- FAX: (650) 506-7227 Attn: Server Technologies Documentation Manager
- Postal service:
  Oracle Corporation
  Server Technologies Documentation
  500 Oracle Parkway, Mailstop 4op11
  Redwood Shores, CA 94065
  USA

If you would like a reply, please give your name, address, telephone number, and (optionally) electronic mail address.

If you have problems with the software, please contact your local Oracle Support Services.

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.

Organization

This document contains:

Part I, "Streams Concepts"

Contains chapters that describe conceptual information relating to Streams.

Chapter 1, "Introduction to Streams"

Introduces the major features of Streams and how they can be used.

Chapter 2, "Streams Capture Process"

Contains conceptual information about the Streams capture process. Includes information about logical change records (LCRs), datatypes and types of changes captured, and supplemental logging, along with information about capture process architecture.

Chapter 3, "Streams Staging and Propagation"

Contains conceptual information about staging and propagation in a Streams environment. Includes information about the differences between captured and user-enqueued events, propagation, the differences between transactional and non-transactional queues, and using `SYS.AnyData` queues. Also includes information about queue and propagation architecture.

Chapter 4, "Streams Apply Process"

Contains conceptual information about the Streams apply process. Includes information about event processing with an apply process, considerations for apply changes to tables, conditions for applying DDL changes, and controlling a trigger's firing property, along with information about the oldest SCN for an apply process and apply process architecture.

Chapter 5, "Rules"

Contains conceptual information about rules. Includes information about rule components, rule sets, and privileges related to rules.

Chapter 6, "How Rules Are Used In Streams"

Contains conceptual information about how rules are used in Streams. Includes information about table rules, subset rules, schema rules, and global rules. Also includes information about rule-based transformations.

Chapter 7, "Streams High Availability Environments"

Contains conceptual information about using Streams for high availability environments.

Part II, "Streams Administration"

Contains chapters that describe managing a capture process, staging, propagation, an apply process, rules, rule-based transformations, logical change records (LCRs), and Streams tags.

Chapter 8, "Preparing a Streams Environment"

Contains information about preparing for a Streams environment. Includes instructions for configuring a Streams administrator, setting initialization parameters that are important to Streams, preparing for a capture process, and configuring networking connectivity.

Chapter 9, "Managing a Capture Process"

Contains information about managing a capture process. Includes instructions for creating, starting, stopping, and altering a capture process, as well as other information related to capture process administration.

Chapter 10, "Managing Staging and Propagation"

Contains information about managing staging and propagation of events in a Streams environment. Includes instructions for creating a `SYS.AnyData` queue, and instructions for enabling, disabling, and altering a propagation, as well as other information related to staging, propagation, and messaging.

Chapter 11, "Managing an Apply Process"

Contains information about managing an apply process. Includes instructions for creating, starting, stopping, and altering an apply process, as well as instructions about using apply process handlers, configuring conflict resolution, and managing an exception queue.

Chapter 12, "Managing Rules and Rule-Based Transformations"

Contains information about managing rules and rule-based transformations. Includes instructions for managing rules and rule sets, as well as information about granting and revoking privileges related to rules. In addition, this chapter includes instructions for creating, altering, and removing rule-based transformations.

Chapter 13, "Other Streams Management Tasks"

Contains information about managing logical change records (LCRs) and Streams tags. Includes instructions for constructing and enqueuing LCRs, and instructions for setting and removing tag values for a session or an apply process.

Chapter 14, "Monitoring a Streams Environment"

Contains information about using data dictionary views and scripts to monitor a Streams environment. Includes information about monitoring capture processes, queues, propagations, apply processes, rules, rule-based transformations, and tags.

Chapter 15, "Troubleshooting a Streams Environment"

Contains information about possible problems in a Streams environment and how to resolve them. Includes information about troubleshooting a capture process, propagation, apply process, messaging client, and the rules used by these Streams clients, as well as information about checking trace files and the alert log for problems.

Part III, "Example Environments and Applications"
Contains chapters that illustrate example environments.

Chapter 16, "Single Database Capture and Apply Example"
Contains a step by step example that configures a single database capture and apply example using Streams. Specifically, this chapter illustrates an example of a single database that captures changes to a table, uses a DML handler during apply to re-enqueue the captured changes into a queue, and then applies a subset of the changes to a different table.

Chapter 17, "Rule-Based Application Example"
Contains step by step examples that illustrate a rule-based application that uses the Oracle rules engine.

Part IV, "Appendixes"
Contains one appendix that describes the XML schema for logical change records (LCRs).

Appendix A, "XML Schema for LCRs"
Contains the definition of the XML schema for LCRs.

Appendix B, "Online Database Upgrade and Maintenance With Streams"
Contains information about performing certain maintenance operations on an Oracle database with little or no down time. These maintenance operations include upgrading to a new version of the Oracle Database, migrating an Oracle Database to a different operating system or character set, upgrading user-created applications, and applying Oracle Database patches.

Related Documentation

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*
- *PL/SQL Packages and Types Reference*

- *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

You may find more information about a particular topic in the other documents in the Oracle documentation set.

Oracle error message documentation is only available in HTML. If you only have access to the Oracle Documentation CD, you can browse the error messages by range. Once you find the specific range, use your browser's "find in page" feature to locate the specific message. When connected to the Internet, you can search for a specific error message using the error message search feature of the Oracle online documentation.

Many of the examples in this book use the sample schemas of the seed database, which is installed by default when you install Oracle. 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://otn.oracle.com/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://otn.oracle.com/documentation/`

In addition, you can find resources related to Oracle Streams at

`http://otn.oracle.com/products/dataint/content.html`

Conventions

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

- Conventions in Text
- Conventions in Code Examples

Conventions in Text

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

Convention	Meaning	Example
Bold	Bold typeface indicates terms that are defined in the text or terms that appear in a glossary, or both.	When you specify this clause, you create an **index-organized table**.
Italics	Italic typeface indicates book titles or emphasis.	*Oracle Database Concepts* Ensure that the recovery catalog and target database do *not* reside on the same disk.
`UPPERCASE monospace (fixed-width) font`	Uppercase monospace typeface indicates elements supplied by the system. Such elements include parameters, privileges, datatypes, RMAN keywords, SQL keywords, SQL*Plus or utility commands, packages and methods, as well as system-supplied column names, database objects and structures, usernames, and roles.	You can specify this clause only for a `NUMBER` column. You can back up the database by using the `BACKUP` command. Query the `TABLE_NAME` column in the `USER_TABLES` data dictionary view. Use the `DBMS_STATS.GENERATE_STATS` procedure.

Convention	Meaning	Example
`lowercase monospace (fixed-width) font`	Lowercase monospace typeface indicates executables, filenames, directory names, and sample user-supplied elements. Such elements include computer and database names, net service names, and connect identifiers, as well as user-supplied database objects and structures, column names, packages and classes, usernames and roles, program units, and parameter values. **Note:** Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.	Enter `sqlplus` to open SQL*Plus. The password is specified in the `orapwd` file. Back up the datafiles and control files in the `/disk1/oracle/dbs` directory. The `department_id`, `department_name`, and `location_id` columns are in the `hr.departments` table. Set the `QUERY_REWRITE_ENABLED` initialization parameter to `true`. Connect as `oe` user. The `JRepUtil` class implements these methods.
`lowercase italic monospace (fixed-width) font`	Lowercase italic monospace font represents placeholders or variables.	You can specify the *`parallel_clause`*. Run `U`*`old_release`*`.SQL` where *`old_release`* refers to the release you installed prior to upgrading.

Conventions in Code Examples

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

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

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

Convention	Meaning	Example
[ ]	Brackets enclose one or more optional items. Do not enter the brackets.	`DECIMAL (`*`digits`*` [ , `*`precision`*` ])`
{ }	Braces enclose two or more items, one of which is required. Do not enter the braces.	`{ENABLE | DISABLE}`

|	A vertical bar represents a choice of two or more options within brackets or braces. Enter one of the options. Do not enter the vertical bar.	`{ENABLE | DISABLE}`
		`[COMPRESS | NOCOMPRESS]`

Convention	Meaning	Example

`. . .`	Horizontal ellipsis points indicate either:
	■ That we have omitted parts of the code that are not directly related to the example
	■ That you can repeat a portion of the code
		`CREATE TABLE ... AS` *`subquery`*`;`
		
		`SELECT` *`col1`*`,` *`col2`*`, ... ,` *`coln`* `FROM employees;`

.
.
.
	Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.	`SQL> SELECT NAME FROM V$DATAFILE;`
		`NAME`
		`------------------------------------`
		`/fsl/dbs/tbs_01.dbf`
		`/fs1/dbs/tbs_02.dbf`
		`.`
		`.`
		`.`
		`/fsl/dbs/tbs_09.dbf`
		`9 rows selected.`

Other notation	You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as shown.	`acctbal NUMBER(11,2);`
		`acct    CONSTANT NUMBER(4) := 3;`

`Italics`	Italicized text indicates placeholders or variables for which you must supply particular values.	`CONNECT SYSTEM/`*`system_password`*
		`DB_NAME =` *`database_name`*

`UPPERCASE`	Uppercase typeface indicates elements supplied by the system. We show these terms in uppercase in order to distinguish them from terms you define. Unless terms appear in brackets, enter them in the order and with the spelling shown. However, because these terms are not case sensitive, you can enter them in lowercase.	`SELECT last_name, employee_id FROM employees;`
		`SELECT * FROM USER_TABLES;`
		`DROP TABLE hr.employees;`

`lowercase`	Lowercase typeface indicates programmatic elements that you supply. For example, lowercase indicates names of tables, columns, or files.
	
	Note: Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.
		`SELECT last_name, employee_id FROM employees;`
		`sqlplus hr/hr`
		`CREATE USER mjones IDENTIFIED BY ty3MU9;`

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. 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 additional information, visit the Oracle Accessibility Program Web site at

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

Accessibility of Code Examples in Documentation JAWS, a Windows screen reader, 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, JAWS 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.

What's New in Oracle Streams?

This section describes new features of Oracle Streams for Oracle Database 10*g* Release 1 (10.1) and provides pointers to additional information where appropriate.

The following sections describe the new features in Oracle Streams:

- 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 (10.1) 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 `SYS.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 (10.1):

- 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 (RAC)
- Support for Function-Based Indexes and Descending Indexes
- Simpler Removal of Rule Sets When a Streams Client Is Dropped

- Simpler Removal of SYS.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 processes, propagations, apply processes, and messaging clients, can use two rule sets: 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 2-19
> - "Creating a Capture Process" on page 9-2

Subset Rules for Capture and Propagation

You can use subset rules for capture processes, propagations and messaging clients, as well as for apply processes.

> **See Also:** "Subset Rules" on page 6-23

Streams Pool

In a single database, you can specify that Streams memory be allocated from a new pool in the SGA called the Streams pool. To configure the Streams pool, specify the size of the pool in bytes using the `STREAMS_POOL_SIZE` initialization parameter.

The Streams pool contains buffered queues and is used for internal communications during parallel capture and apply.

See Also:

- "Buffered Queues" on page 3-16
- "Setting Initialization Parameters Relevant to Streams" on page 8-6

Access To Buffered Queue Information

The following new dynamic performance views enable you to monitor buffered queues:

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

See Also:

- "Buffered Queues" on page 3-16
- *Oracle Streams Replication Administrator's Guide* for information about monitoring buffered queues

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 rules in the `DBMS_STREAMS_ADM` package include an `and_condition` parameter. This parameter enables you to add custom conditions to system-created rules.

See Also: "System-Created Rules with Added User-Defined Conditions" on page 6-44

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 transformations.

See Also:

- "Rule-Based Transformations" on page 6-63
- "Managing Rule-Based Transformations" on page 12-18

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 events that satisfy a particular rule. When an event satisfies such a rule in an apply process rule set, the apply process enqueues the event into the specified queue.

See Also: "Specifying Event Enqueues by Apply Processes" on page 11-21

Execution Directives Upon Apply

A new procedure, SET_EXECUTE in the DBMS_APPLY_ADM package, enables you to specify that apply processes do not execute events that satisfy a certain rule.

See Also: "Specifying Execute Directives for Apply Processes" on page 11-23

Support for Additional Datatypes

Streams capture processes and apply processes now support the following additional datatypes:

- NCLOB
- BINARY_FLOAT
- BINARY_DOUBLE
- LONG
- LONG RAW

Logical change records (LCRs) containing these datatypes may also be propagated using propagations.

See Also:

- "Datatypes Captured" on page 2-8
- "Datatypes Applied" on page 4-11

Support for Index-Organized Tables

Streams capture processes and apply processes now support processing changes to index-organized tables.

See Also:

- "Types of DML Changes Captured" on page 2-10
- *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 Events Using Precommit Handlers" on page 4-8
- "Managing the Precommit Handler for an Apply Process" on page 11-18

Better Interoperation With Oracle Real Application Clusters (RAC)

The following are specific enhancements that improve Streams interoperation with RAC:

- Streams capture processes 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 2-27
- "Streams Apply Processes and Oracle Real Application Clusters" on page 4-13

Support for Function-Based Indexes and Descending Indexes

Streams capture processes and apply processes 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 sets, 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 processes, propagations, apply processes, and messaging clients. If this procedure drops a rule set, then this procedure also drops any rules in the rule set that are not in another rule set.

See Also:

- "Dropping a Capture Process" on page 9-26
- "Dropping a Propagation" on page 10-19
- "Dropping an Apply Process" on page 11-10
- *PL/SQL Packages and Types Reference* for more information about the procedures for dropping Streams clients

Simpler Removal of SYS.AnyData Queues

A new procedure, `REMOVE_QUEUE` in the `DBMS_STREAMS_ADM` package, enables you to remove a `SYS.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 a SYS.AnyData Queue" on page 10-6
- *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 2-32
- "First SCN and Start SCN" on page 2-24
- "First SCN and Start SCN Specifications During Capture Process Creation" on page 2-40

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 14, "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: *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 8-2

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: *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 (10.1):

- 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 Whether Old Values Are Compared for 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 instantiations.

> **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 instantiations
- *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 conditions, rule-based transformations, apply handlers, error handlers, 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 events from the error queue (`ERROR_EXECUTION` type) instead of the apply process queue (`APPLY` type).

See Also:

- "Managing an Error Handler" on page 11-25 for an example of an error handler that uses the `GET_STREAMS_NAME` function
- *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*
- *PL/SQL Packages and Types Reference*

Control Whether Old Values Are Compared for 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: *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 2-6

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 processes 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:

- Access to the commit SCN of LCRs

  The GET_COMMIT_SCN member function returns the commit SCN of the transaction to which the current LCR belongs.

- The ability to get and set extra attributes in LCRs

  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.

- Access to compatibility information for LCRs

  The GET_COMPATIBLE member function returns the minimal database compatibility required to support an LCR.

- The ability to convert LONG data in a row LCR into a LOB chunk

  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:

  - *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 6-56 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 (10.1):

- 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 messages from a `SYS.AnyData` queue based on rules. 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 6-36
> - "Configuring a Messaging Client and Message Notification" on page 10-25
> - *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 messages into and dequeuing messages from a `SYS.AnyData` queue.

See Also:

- "Configuring a Messaging Client and Message Notification" on page 10-25
- *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 clients and apply processes, and it enables you to create the rules for user-enqueued messages that control the behavior of these messaging clients and apply processes.

See Also:

- "Message Rules" on page 6-36
- *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 propagations and create rules for propagations that propagate user-enqueued messages.

See Also: *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 10-25
- *PL/SQL Packages and Types Reference* for more information about the SET_MESSAGE_NOTIFICATION procedure

Rules Interface Enhancements

The following are rules interface enhancements for Oracle Database 10*g* Release 1 (10.1):

- 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 5-13
- Chapter 17, "Rule-Based Application Example" for examples that use iterative evaluation results
- *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 sets and rule evaluations:

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

See Also: "Monitoring Rules and Rule-Based Transformations" on page 14-56

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*.

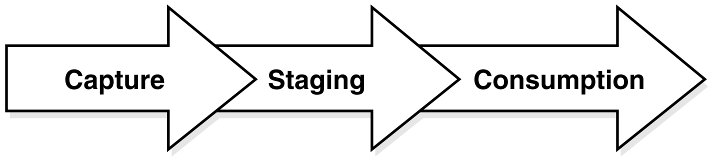
This chapter contains these topics:

Overview of Streams

Oracle Streams enables information sharing. Using Oracle Streams, you can share data and events 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 events, and sharing the events 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 events 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 events in the database automatically, including, but not limited to, data manipulation language (DML) changes and data definition language (DDL) changes. You also can put user-defined events into a stream, and Streams can propagate the information to other databases or applications automatically. When events 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 Events at a Database

A **capture process** can capture 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 an event called a logical change record (**LCR**). Rules determine which changes are captured by a capture process, and these captured changes are called **captured events**.

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

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

Stage Events in a Queue

Events are stored (or **staged**) in a queue. These events may be captured events or user-enqueued events. A capture process enqueues events into a **`SYS.AnyData` queue**. A a `SYS.AnyData` queue can stage events of different types. Users and applications may enqueue events into a `SYS.AnyData` queue or into a **typed queue**. A typed queue can stage events of one specific type only.

Propagate Events From One Queue To Another

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

Consume Events

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

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

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

Other Capabilities of Streams

Other capabilities of Streams include the following:

- Directed networks
- Automatic conflict detection and resolution
- Transformations
- 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

Oracle Streams Advanced Queuing (AQ) enables user applications to enqueue messages 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 may be configured to stage messages of a particular type only, or a queue may be configured as a `SYS.AnyData` queue. Messages of almost any type can be wrapped in a `SYS.AnyData` wrapper and staged in `SYS.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 clients can dequeue messages directly from this queue. If an application is remote, then a queue may 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 databases and then applied by Streams apply processes.

The destination databases can allow DML and DDL changes to the same database objects, and these changes may or may 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 may enqueue events into a queue explicitly, or a Streams capture process may capture a database event. These captured events may be DML or DDL changes. Propagations may propagate events in a stream through multiple queues. Finally, a user application may dequeue events explicitly, or a Streams apply process may dequeue events implicitly. An apply process may re-enqueue these events 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.

Capture processes, propagations, apply processes, and messaging clients perform actions based on rules. 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. Events staged in a queue may be dequeued explicitly by a messaging client or an application, and then actions can be taken based on these events, which may 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 10, "Managing Staging and Propagation", and *Oracle Streams Advanced Queuing User's Guide and Reference* for more information about explicitly enqueuing and dequeuing events
- Chapter 16, "Single Database Capture and Apply Example" for an example environment that explicitly dequeues events

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 log information 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 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 may include specialized indexes or other database objects. Therefore, these standby databases can be queried as updates are applied, making Oracle Data Guard a good solution for off loading queries from a production database.

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.

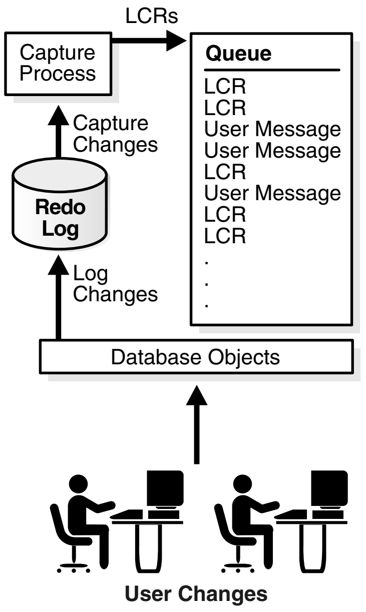
See Also:

- Chapter 7, "Streams High Availability Environments"
- *Oracle Data Guard Concepts and Administration* for more information about logical standby databases

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 events 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



Note: 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.

You can configure change capture locally at a source database or remotely at a **downstream database**. If a capture process runs on a downstream database, then redo log files from the source database are copied to the downstream database, and the capture process captures changes in these redo log files.

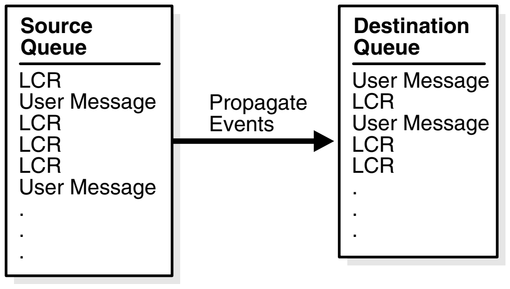
> **See Also:** Chapter 2, "Streams Capture Process" for more information about capture processes and for detailed information about which DML and DDL statements are captured by a capture process

Overview of Event Staging and Propagation

Streams uses queues to stage events for propagation or consumption. Propagations send events from one queue to another, and these queues can be in the same database or in different databases. The queue from which the events are propagated is called the **source queue**, and the queue that receives the events 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.

Events that are staged in a queue can be consumed by an apply process, a messaging client, or by an application. Rules determine which events 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

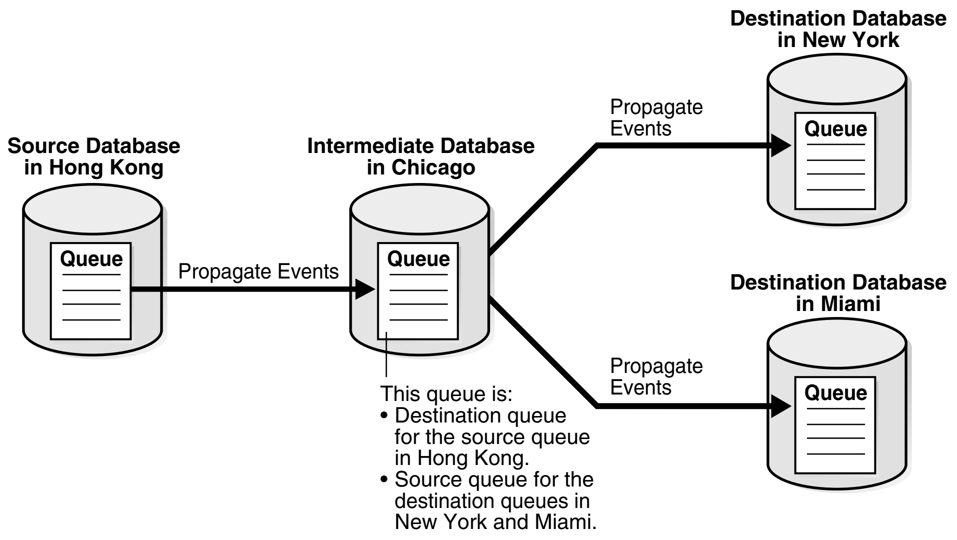


Overview of Directed Networks

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

Figure 1–4 shows an example directed networks environment. Notice that, in this example, the queue at the intermediate database in Chicago is both a source queue and a destination queue.

Figure 1–4 Example Directed Networks Environment

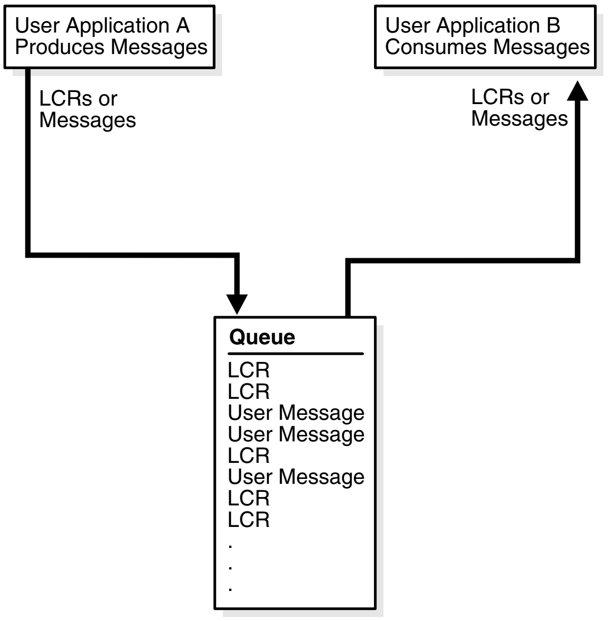

Destination Database in New York
Propagate Events
Queue
Source Database in Hong Kong
Intermediate Database in Chicago
Queue
Propagate Events
Queue
Destination Database in Miami
Propagate Events
Queue
This queue is:
• Destination queue for the source queue in Hong Kong.
• Source queue for the destination queues in New York and Miami.

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

Explicit Enqueue and Dequeue of Events

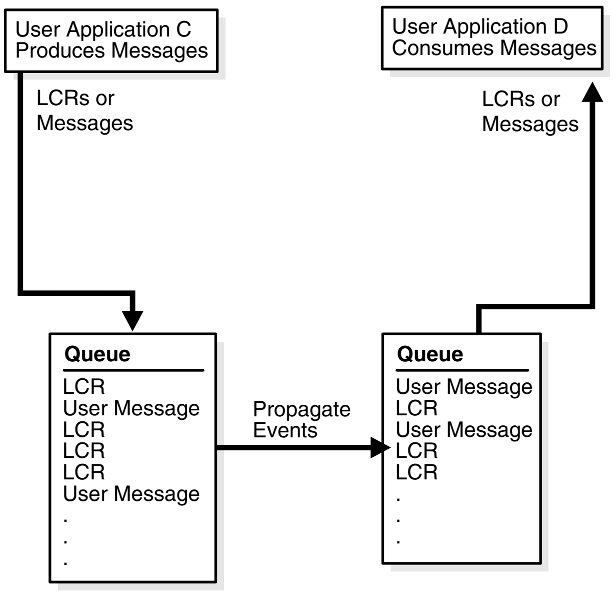
User applications can enqueue events into a queue explicitly. The user applications can format these user-enqueued events as LCRs or user messages, and an apply process, a messaging client, or a user application can consume these events. Events that were enqueued explicitly into a queue can be propagated to another queue or explicitly dequeued from the same queue. Figure 1–5 shows explicit enqueue of events into and dequeue of events from the same queue.

Figure 1–5 Explicit Enqueue and Dequeue of Events in a Single Queue



When events are propagated between queues, events that were enqueued explicitly into a source queue can be dequeued explicitly from a destination queue by a messaging client or user application. These events also may be processed by an apply process. Figure 1–6 shows explicit enqueue of events into a source queue, propagation to a destination queue, and then explicit dequeue of events from the destination queue.

Figure 1–6 Explicit Enqueue, Propagation, and Dequeue of Events



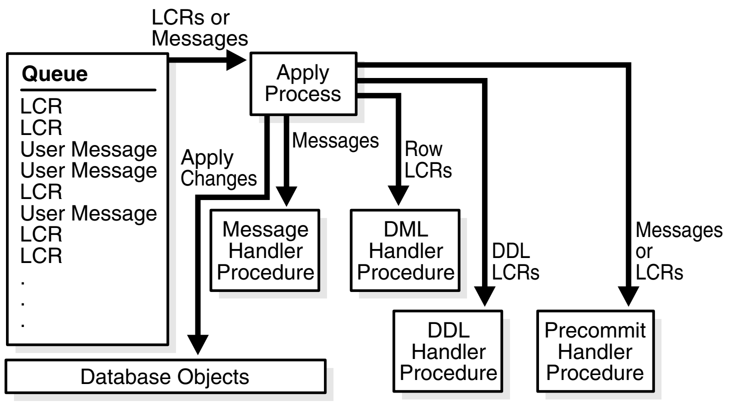
See Also: "SYS.AnyData Queues and User Messages" on page 3-12 for more information about explicit enqueue and dequeue of events

Overview of the Apply Process

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

Typically, an apply process applies events to the local database where it is running, but, in a heterogeneous database environment, it can be configured to apply events at a remote non-Oracle database. Rules determine which events are dequeued by an apply process. Figure 1–7 shows an apply process processing LCRs and user messages.

Figure 1–7 Apply Process

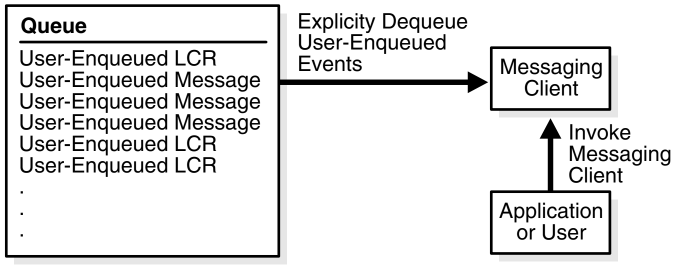


See Also: Chapter 4, "Streams Apply Process"

Overview of the Messaging Client

A messaging client consumes user-enqueued events when it is invoked by an application or a user. Rules determine which user-enqueued events are dequeued by a messaging client. These user-enqueued events may be LCRs or user messages. Figure 1–8 shows a messaging client dequeuing user-enqueued events.

Figure 1–8 Messaging Client



See Also: "Messaging Clients" on page 3-11

Overview of Automatic Conflict Detection and Resolution

An apply process detects conflicts automatically when directly applying LCRs in a replication environment. 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 Oracle's 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 events 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 expressions and conditions and returns a Boolean value, which is a value of `TRUE`, `FALSE`, or `NULL` (unknown), based on an event.
- The **rule evaluation context** defines external data that can be referenced in rule conditions. The external data either can exist as external variables, as table data, or both.
- The **rule 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 may be positive or negative.

For example, the following rule condition may 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 may 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 events with a capture process, propagating events with a propagation, applying events with an apply process, dequeuing events with a messaging client, and discarding events.

See Also:

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

Overview of Transformations

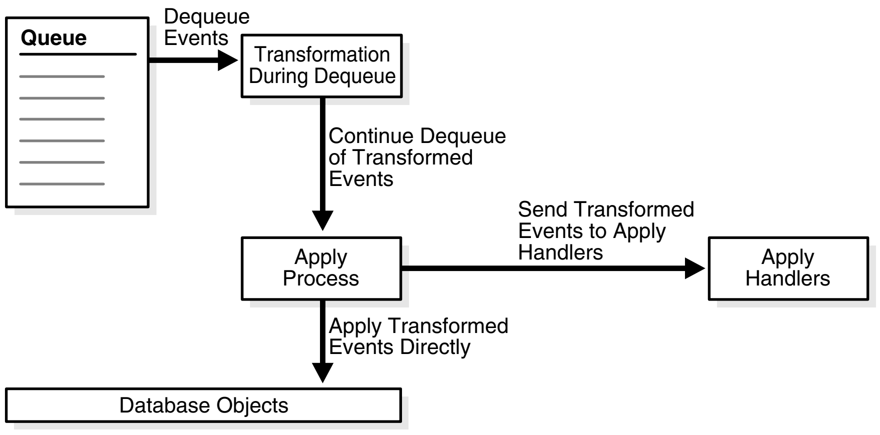
A **rule-based transformation** is any modification to an event that results when a rule in a positive rule set evaluates to `TRUE`. For example, a rule-based transformation can change the datatype of a particular column in a table for an event. In this case, the transformation can be a PL/SQL function that takes as input a `SYS.AnyData` object containing an LCR with a `NUMBER` datatype for a column and returns a `SYS.AnyData` object containing an LCR with a `VARCHAR2` datatype for the same column.

A transformation can occur at the following times:

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

When a transformation is performed during apply, an apply process may apply the transformed event directly or send the transformed event to an apply handler for processing. Figure 1–9 shows a rule-based transformation during apply.

Figure 1–9 Transformation During Apply


Queue
Dequeue Events
Transformation During Dequeue
Continue Dequeue of Transformed Events
Send Transformed Events to Apply Handlers
Apply Process
Apply Handlers
Apply Transformed Events Directly
Database Objects

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 processes, propagations, apply processes, and messaging clients.

See Also: "Rule-Based Transformations" on page 6-63

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, rules may have conditions relating to tag values to control the behavior of Streams clients. 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 databases for each LCR. Tags may 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

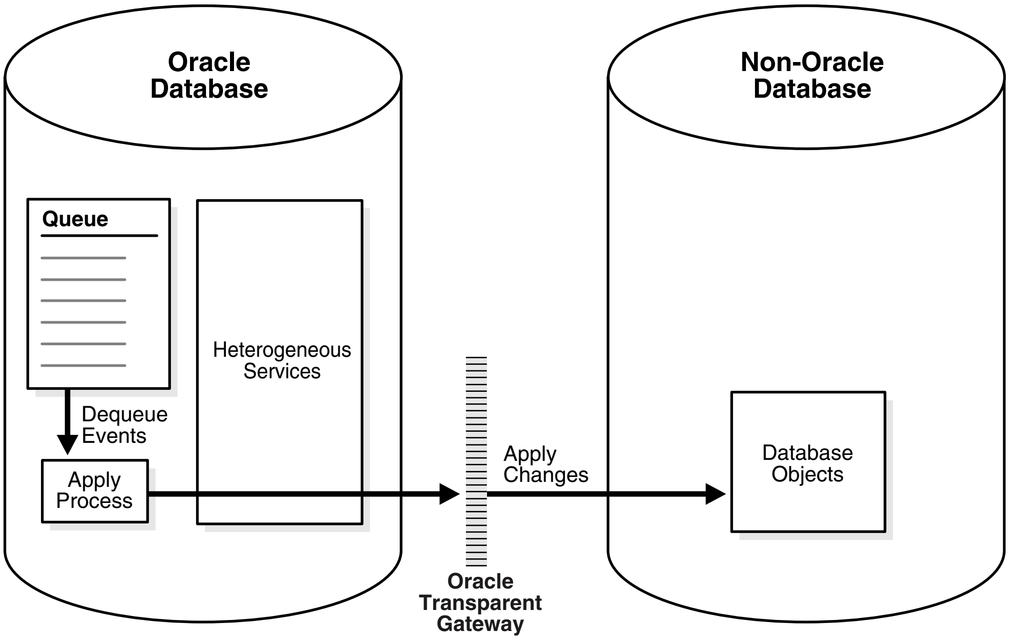
Overview of Oracle to Non-Oracle Data Sharing

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 events
- An apply process to dequeue and apply events

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 events 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 events to the non-Oracle database across a network connection through a gateway. Figure 1–10 shows an Oracle databases sharing data with a non-Oracle database.

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

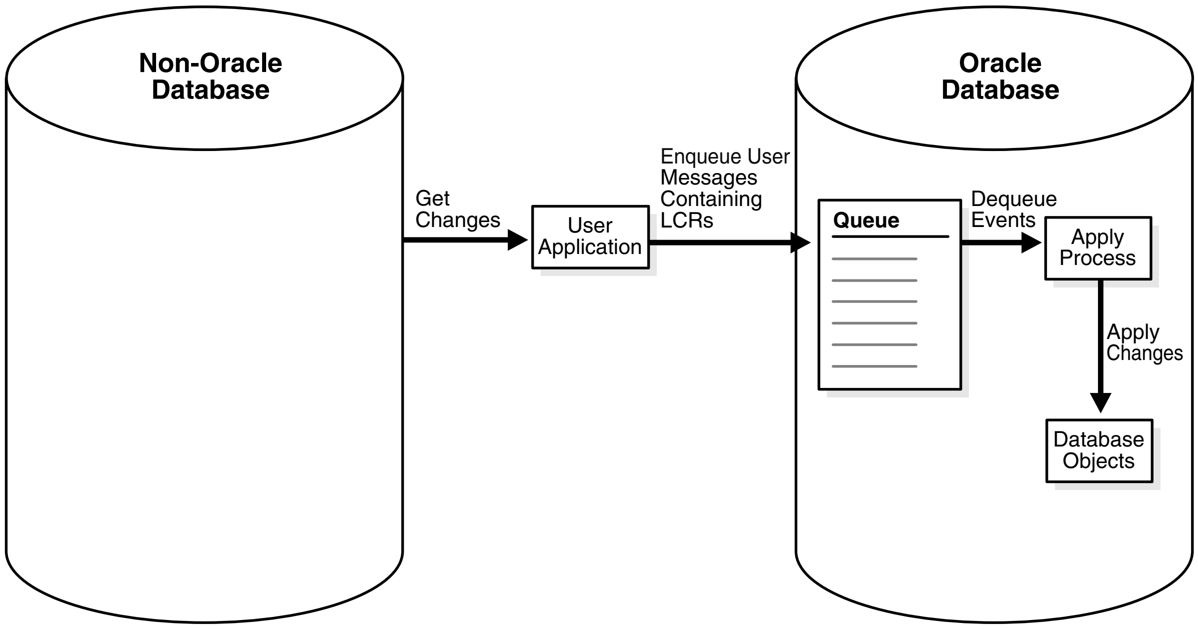


See Also: *Oracle Database Heterogeneous Connectivity Administrator's Guide* for more information about Heterogeneous Services

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 1–11 shows a non-Oracle databases sharing data with an Oracle database.

Figure 1–11 Non-Oracle to Oracle Heterogeneous Data Sharing



Example Streams Configurations

Figure 1–12 shows how Streams might be configured to share information within a single database, while Figure 1–13 shows how Streams might be configured to share information between two different databases.

Figure 1–12 Streams Configuration in a Single Database

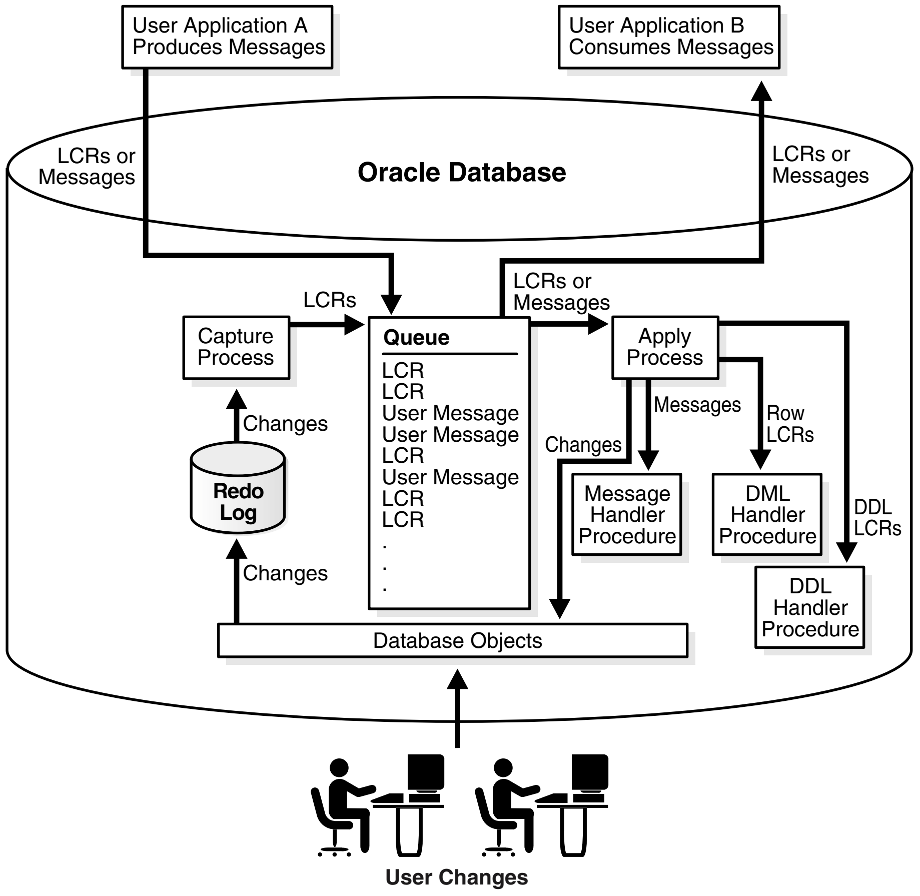
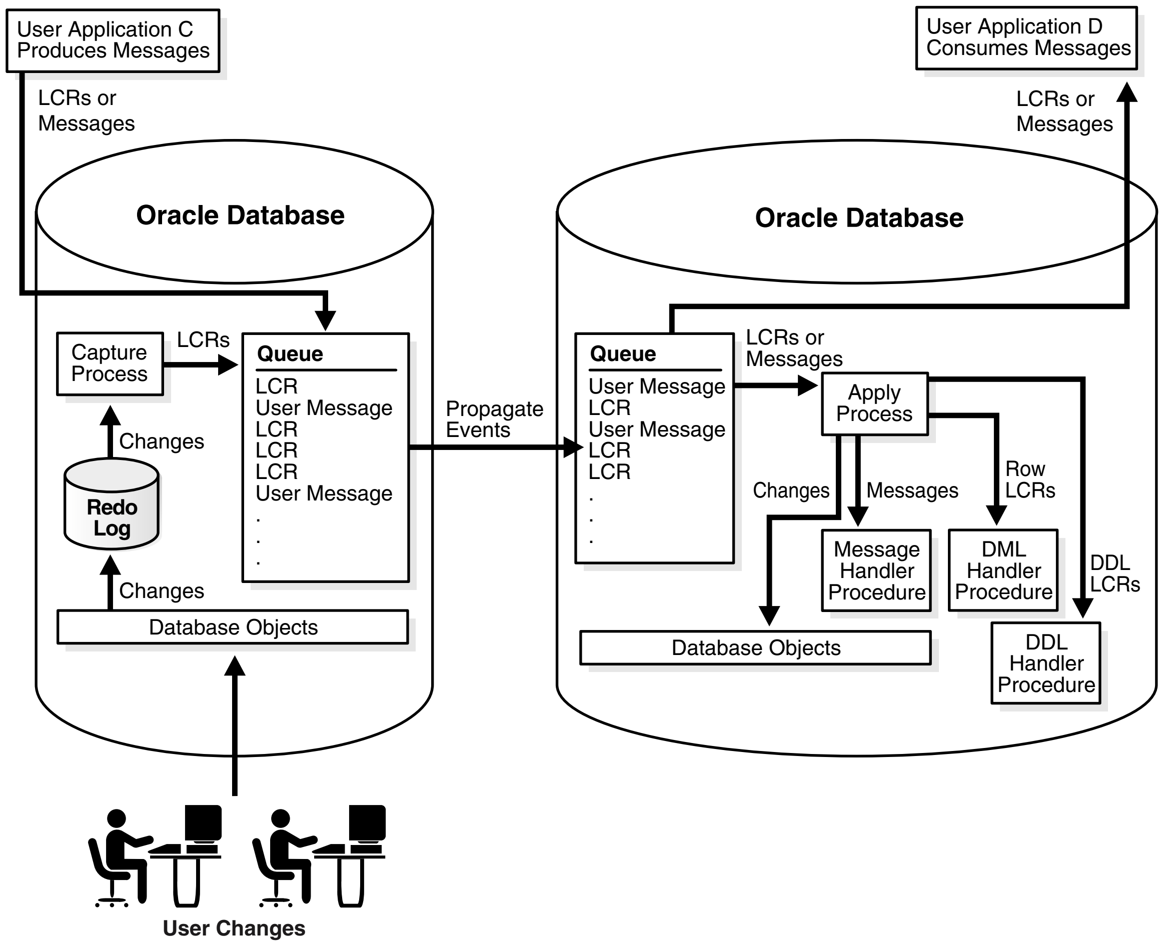


Figure 1–13 Streams Configuration Sharing Information Between Databases



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 tool, while the Streams tool in the Oracle Enterprise Manager Console 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: *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 rules for capture processes, propagations, and apply processes at the table, schema, and database level. This package also enables you to add rules that control which events a propagation propagates and which events a messaging client dequeues. This package also contains procedures for creating queues 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 for specific tablespaces. 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 handlers, set enqueue destinations for events, and specify execution directives for events. 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 a SYS.AnyData queue.

DBMS_RULE_ADM Package

The DBMS_RULE_ADM package provides an administrative interface for creating and managing rules, rule sets, and rule evaluation contexts. 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 SYS.AnyData objects into LCR objects, to return information about Streams attributes and Streams clients, and to annotate redo entries generated by a session with a tag. This tag may affect the behavior of a capture process, a propagation job, an apply process, or a messaging client whose rules include specifications for these tags in redo entries or LCRs.

DBMS_STREAMS_AUTH Package

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

DBMS_STREAMS_TABLESPACE_ADM

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.

Streams Data Dictionary Views

Every database in a Streams environment has Streams data dictionary views. These views maintain administrative information about local rules, objects, capture processes, propagations, apply processes, and messaging clients. You can use these views to monitor your Streams environment.

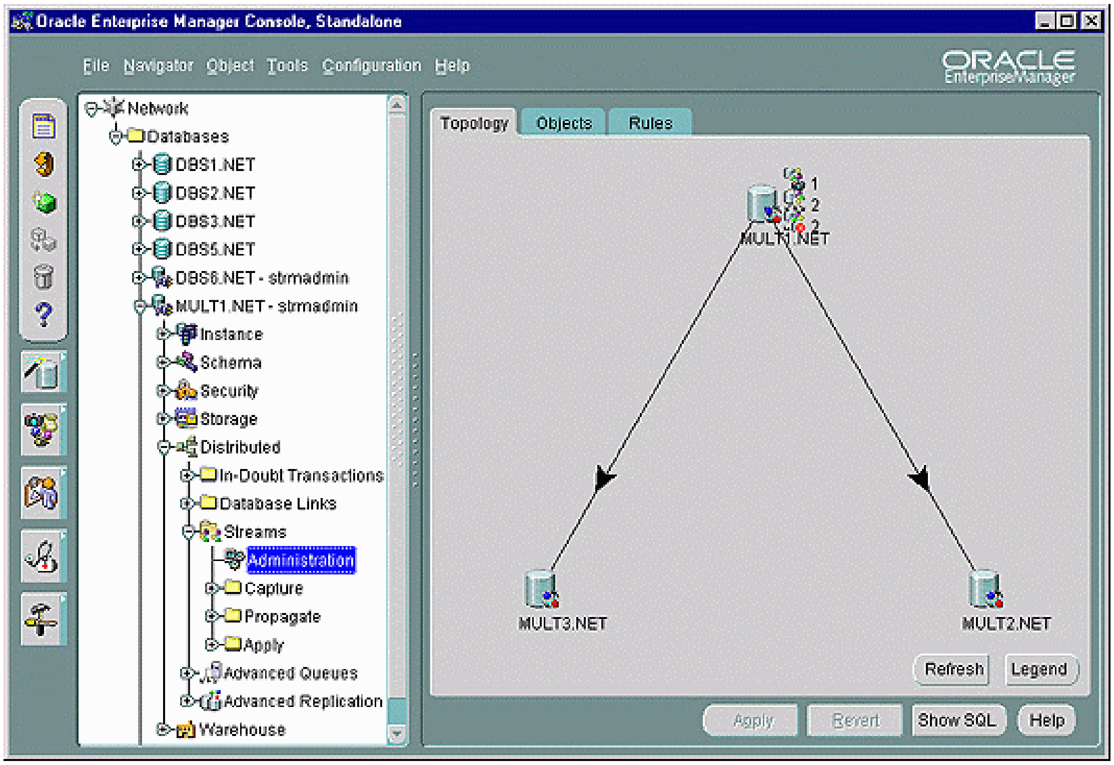
See Also:

- Chapter 14, "Monitoring a Streams Environment"
- *Oracle Streams Replication Administrator's Guide* for example 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 also can 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 is the primary documentation source for this tool. Figure 1–14 shows the **Topology** tab in the Streams tool.

Figure 1–14 Streams Tool



See Also: See the online help for the Streams tool in the Oracle Enterprise Manager Console for more information about using it

2 Streams Capture Process

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

This chapter contains these topics:

See Also: Chapter 9, "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's 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 may 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 remote database is called the **downstream database**. If a capture process runs on a downstream database, then archived redo log files from the source database are copied to the downstream database, and the capture process captures changes from these files at the downstream database.

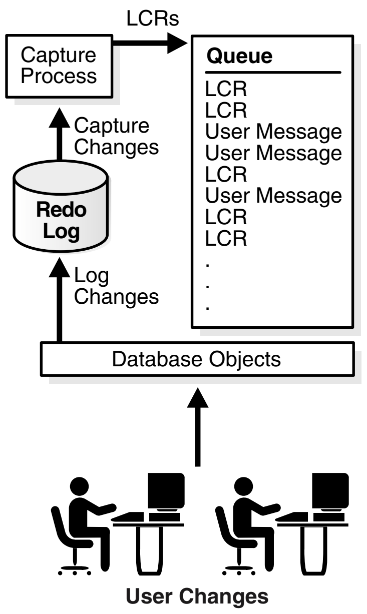
Logical Change Records (LCRs)

A capture process reformats changes captured from the redo log into LCRs. An LCR is an object 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 an event containing the LCR into a queue. A capture process is always associated with a single `SYS.AnyData` queue, and it enqueues events into this queue only. For improved performance, captured events always are stored in a buffered queue, which is System Global Area (SGA) memory associated with a `SYS.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 a `SYS.AnyData` queue, not with a typed queue.

Figure 2–1 The Capture Process



See Also:

- *Oracle Streams Replication Administrator's Guide* for information about managing LCRs
- *PL/SQL Packages and Types Reference* for more information about LCR types
- "Buffered Queues" on page 3-16

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 may insert or merge multiple rows into a table, may update multiple rows in a table, or may 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 may 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 also may 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 may 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 LCRs will be propagated by a propagation or applied by an apply process, then, to avoid propagation and apply problems, Oracle Corporation 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 may 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, nor in 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 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 LCRs" on page 9-38
- "Viewing the Extra Attributes Captured by Each Capture Process" on page 14-17
- *PL/SQL Packages and Types Reference* for more information about the `INCLUDE_EXTRA_ATTRIBUTE` procedure

Capture Process Rules

A capture process either captures or discards changes based on rules 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 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). A capture process raises an error if it tries to create a row LCR for a DML change to a table containing 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 rules used by the capture process to avoid the error, and restart the capture process.

Note:

- You may 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 may create a row LCR for the change and continue to process it. In this case, a change that includes an unsupported datatype may cause the capture process to raise an error, even if the change does not satisfy the rule sets used by the capture process. The `DBMS_STREAMS_ADM` package creates only simple rules.
- Some of the datatypes listed previously in this section may 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:

- Chapter 6, "How Rules Are Used In Streams" for more information about rule sets for Streams clients and for information about how events satisfy rule sets
- "Capture Process Rule Evaluation" on page 2-49
- "Datatypes Applied" on page 4-11 for information about the datatypes that can be applied by an apply process
- *Oracle Database SQL Reference* for more information about these 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 meets the following conditions:
  - The index-organized table does not require an `OVERFLOW` clause.
  - The index-organized table does not contain any columns of the following datatypes: `LONG`, `LONG RAW`, `CLOB`, `NCLOB`, `BLOB`, `BFILE`, `ROWID`, `UROWID`, and user-defined types (including object types, `REF`s, varrays, and nested tables).
  - If the index-organized table is partitioned, then it does not have row movement enabled.

  If an index-organized table does not meet these requirements, 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 sets.
- A capture process does not capture `CALL`, `EXPLAIN PLAN`, or `LOCK TABLE` statements.
- A capture process cannot capture DML changes made to temporary tables or object tables.
- If you share a sequence at multiple databases, then sequence values used for individual rows at these databases may 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 2-8 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 clients and for information about how events 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 use to avoid having the same sequence-generated value for two different rows at different databases

Types of DDL Changes Ignored by a Capture Process

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

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

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.

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 clients and for information about how events 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 may be captured by a capture process if the changes satisfy the capture process rule sets.

In addition, online table redefinition using the DBMS_REDEFINITION package is not supported on a table or schema for which a capture process captures changes.

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 Corporation 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 databases of the source database, then apply errors may 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 propagations can propagate the changes to the destination databases, but these objects may not exist at the destination databases, or, the objects may exist at the destination database, but the rows related to these changes may not exist.

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 may also be required in environments where changes are not applied to database objects directly by an apply process. In such environments, an apply handler may process the changes without applying them to the database objects, and the supplemental information may 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 rules 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 may be set for these objects automatically. Instantiation is required whenever an apply process dequeues captured LCRs, 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. The following sections describe these options in detail.

Local Capture

Local capture means that a capture process runs on the source database. Figure 2–1 on page 2-3 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 may 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 may use this existing LogMiner data dictionary, or they may create new LogMiner data dictionaries.
- A capture process scans the redo log for changes using LogMiner.
- The rules engine evaluates changes based on the rules in one or more of the capture process rule sets.
- The capture process enqueues changes that satisfy the rules in its rule sets into a local `SYS.AnyData` queue.
- If the captured changes are shared with one or more other databases, then one or more propagations 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

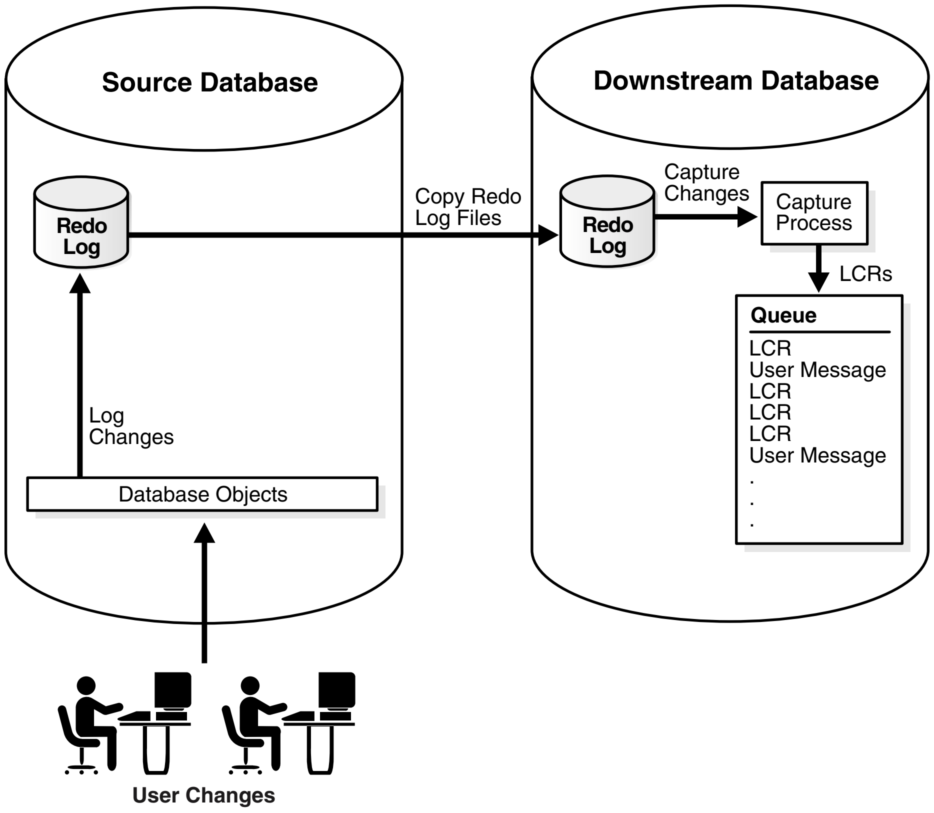
The following are the advantages of using local capture:

- 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 LCRs are propagated to other databases, the captured LCRs may 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 may 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 rule-based transformations are simpler to configure if the capture process is running at the local source database. For example, if you use local capture, then a rule-based transformation may use cached information in a PL/SQL session variable which is populated with data stored at the source database.
- In a Streams environment where events are captured and applied in the same database, it may be simpler, and may use less 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. Archived redo log files from the source database are copied to the downstream database, and the capture process captures changes in these files at the downstream database. You can copy the archived redo log files to the downstream database using log transport services, the `DBMS_FILE_TRANSFER` package, file transfer protocol (FTP), or some other mechanism.

Figure 2–2 Downstream Capture



Note: As illustrated in Figure 2–2, 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.

You can configure multiple capture processes at a downstream database to capture changes from a single source database. You also can copy redo log files from multiple source databases and configure multiple capture processes to capture changes in these redo log files at a single downstream database.

In addition, a single database may 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.

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

The Downstream Database Performs Most Change Capture Actions

If you configure 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 log 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 log files are copied to the downstream database from the source database. Additional downstream capture processes for the same source database may use this existing LogMiner data dictionary, or they may create new LogMiner data dictionaries.
- A capture process scans the redo log files for changes using LogMiner.
- The rules engine evaluates changes based on the rules in one or more of the capture process rule sets.

- The capture process enqueues changes that satisfy the rules in its rule sets into a local `SYS.AnyData` queue.
- If the captured changes are shared with one or more other databases, then one or more propagations propagate these changes 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 may 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 log files must be copied from the computer system running the source database to the computer system running the downstream database. Typically, log transport services copies these 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 clients and for information about how events satisfy rule sets

Advantages of Downstream Capture

The following are the advantages of using downstream capture:

- Capturing changes uses less 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 may be simplified by running multiple 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.

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

Note: If two different capture processes capture changes from a single source database, then the resulting LCRs from these capture processes should never be staged in the same queue if the queue is used by an apply process that applies any of these changes. Instead, a separate queue and apply process should be used to apply changes from each capture process.

Optional Database Link from the Downstream Database to the Source Database

When you create or alter a downstream capture process, you can optionally 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: "Creating a Downstream Capture Process That Uses a Database Link" on page 9-9 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 version of Oracle as the source database or higher.
- 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 ALL_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 event dequeued by the relevant apply processes. All events 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 log information 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 CN does not need to be modified for normal operation of a capture process.

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 may 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 may take some time. Therefore, Oracle Corporation 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 may 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 "Capture Process Creation" on page 2-32 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 may 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 may have been prepared for instantiation before a new capture process is created. For example, if you want to create a new capture process for source database whose changes are already being captured by one or more existing capture processes, then some or all of the database objects may 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 may 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 2-32

Streams Capture Processes and RESTRICTED SESSION

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

When the restricted session is enabled in a running database by the SQL statement `ALTER SYSTEM` with the `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 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 a 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 to all instances in the RAC environment. In a RAC environment, a capture process reads changes made by all instances.

Each capture process is started on the owner instance for its `SYS.AnyData` queue, even if the start 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.

See Also:

- "SYS.AnyData Queues and Oracle Real Application Clusters" on page 3-14 for information about primary and secondary instance ownership for queues
- "Streams Apply Processes and Oracle Real Application Clusters" on page 4-13
- *Oracle Database Reference* for more information about the `DBA_QUEUE_TABLES` data dictionary view
- *Oracle Real Application Clusters Administrator's 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 rules that control which changes a capture process captures.

Changes are captured by a **capture user**. The capture user captures all changes that satisfy the capture process rule sets. In addition, the capture user runs all rule-based transformations 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 rule-based transformation functions specified for rules in the positive rule set, and privileges to enqueue events into the capture process queue. A capture process can be associated with only one user, but one user may be associated with many capture processes.

> **See Also:** "Configuring a Streams Administrator" on page 8-2 for information about the required privileges

This section discusses the following topics:

- Capture Process Components
- Capture Process States
- Multiple Capture Processes in a Single Database
- Capture Process Checkpoints
- Capture Process Creation
- A New First SCN Value and Purged LogMiner Dictionary Information
- The Streams Data Dictionary
- ARCHIVELOG Mode and a Capture Process
- Capture Process Parameters
- Capture Process Rule Evaluation
- Persistent Capture Process Status Upon Database Restart

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 rules 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. The capture process (`c`*`nnn`*) is an Oracle background process.

See Also:

- "Capture Process Parallelism" on page 2-48 for more information about the `parallelism` parameter
- "Capture Process Rule Evaluation" on page 2-49
- *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 sets
- `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 propagations and apply processes are consuming messages slower than the capture process is creating them. This state indicates flow control that is used to reduce spilling of captured LCRs when propagation or apply has fallen behind.
- `SHUTTING DOWN` - Stopping

See Also: "Displaying General Information About Each Capture Process" on page 14-8 for a query that displays the state of a capture process

Multiple Capture Processes in a Single Database

If you run multiple capture processes on 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, you should increase the size of the Streams pool by 10 MB for each capture process parallelism on a database. For example, if you have two capture processes running on 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).

Note:

- Oracle Corporation recommends that each capture process use a separate queue to keep LCRs from different capture processes separate.
- If the size of the Streams pool is zero, then Streams may use up to 10% of the shared pool. The `STREAMS_POOL_SIZE` initialization parameter controls the size of the Streams pool.
- Each capture process uses one LogMiner session.

See Also: "Setting Initialization Parameters Relevant to Streams" on page 8-6 for more information about the `STREAMS_POOL_SIZE` initialization parameter

Capture Process Checkpoints

At regular intervals, a capture process tries to record a **checkpoint**. At a checkpoint, the capture process records its current state persistently in the data dictionary of the database running the capture process.

The **required checkpoint SCN** is the lowest checkpoint SCN for which a capture process requires redo information. 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.

Also, the SCN value 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.

See Also:

- "The LogMiner Data Dictionary for a Capture Process" on page 2-34
- "First SCN and Start SCN Specifications During Capture Process Creation" on page 2-40

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 rules may 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.

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.

Note:

- To create a capture process at a downstream database, you must use the DBMS_CAPTURE_ADM package.
- 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 9, "Managing a Capture Process" and *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 may 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 may share a LogMiner data dictionary, or each capture process may 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 may perform a build of data dictionary information in the redo log multiple times, and a particular build may or may 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 information that a capture process must scan subsequently. Therefore, you should run the `BUILD` procedure only when necessary.

A capture process requires a LogMiner data dictionary because the information in the primary data dictionary may not apply to the changes being captured from the redo log. These changes may 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 may 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 2-49
- "First SCN and Start SCN" on page 2-24
- "Capture Process States" on page 2-30
- *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 either can 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 check point. 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 may 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 log information before it reaches its start SCN.

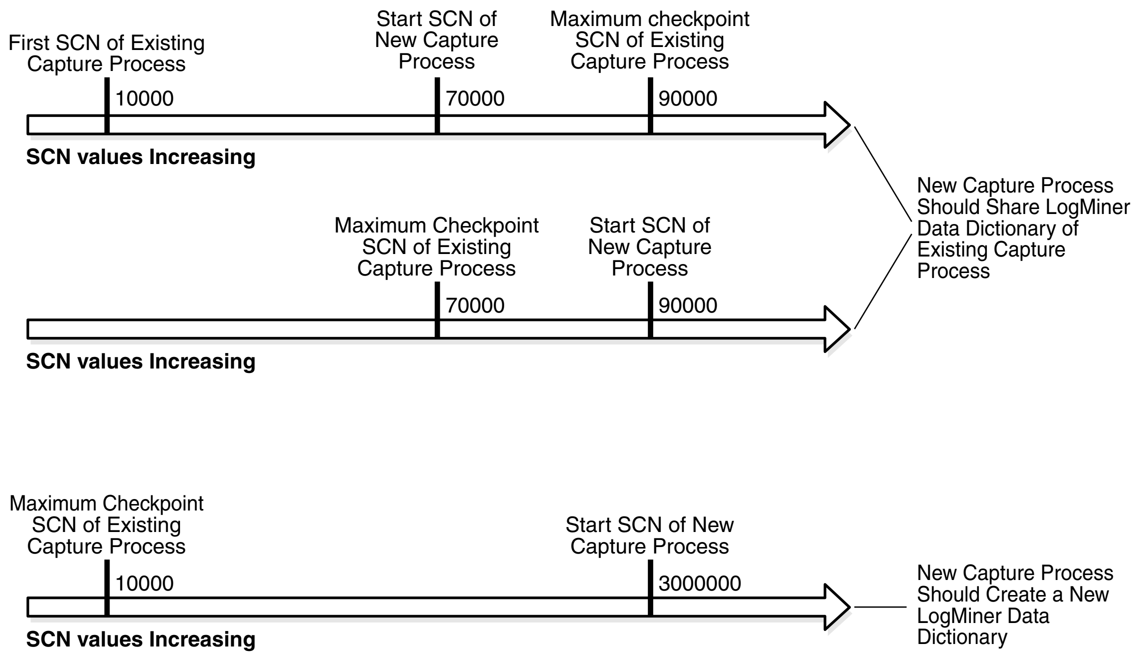
A capture process creates a new LogMiner data dictionary when the `first_scn` parameter is non-`NULL` during capture process creation. Follow these guidelines when you decide if a new capture process should share an existing LogMiner data dictionary or create a new one:

- 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 may 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 may 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 may take a long time for the capture process to catch up to the start SCN. In this case, it may 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 information unnecessarily.

Figure 2–3 illustrates these guidelines.

Figure 2–3 Deciding Whether to Share a LogMiner Data Dictionary


First SCN of Existing Capture Process
10000
Start SCN of New Capture Process
70000
Maximum checkpoint SCN of Existing Capture Process
90000
SCN values Increasing
Maximum Checkpoint SCN of Existing Capture Process
70000
Start SCN of New Capture Process
90000
SCN values Increasing
New Capture Process Should Share LogMiner Data Dictionary of Existing Capture Process
Maximum Checkpoint SCN of Existing Capture Process
10000
Start SCN of New Capture Process
3000000
SCN values Increasing
New Capture Process Should Create a New 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 2-24
- "Capture Process Checkpoints" on page 2-31

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 information 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 information before the start SCN. Oracle Corporation 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 should 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 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 there is at least one existing capture process that 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:

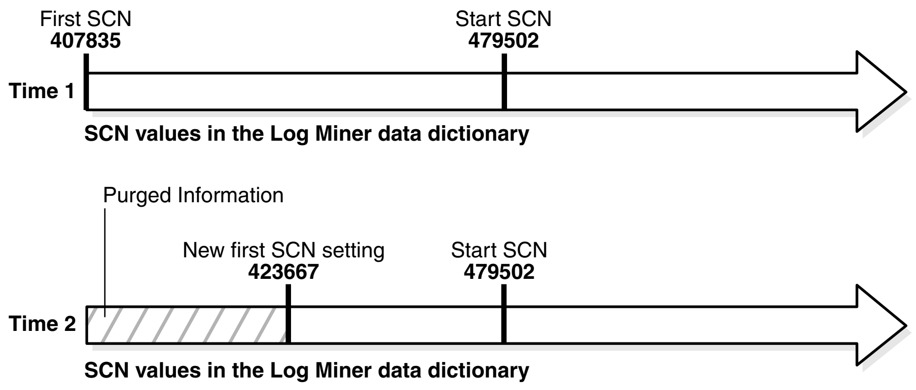
- "NULL First SCN and Non-NULL Start SCN for a Local Capture Process" on page 2-42
- "NULL First SCN and Non-NULL Start SCN for a Downstream Capture Process" on page 2-43

A New First SCN Value and Purged LogMiner 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.

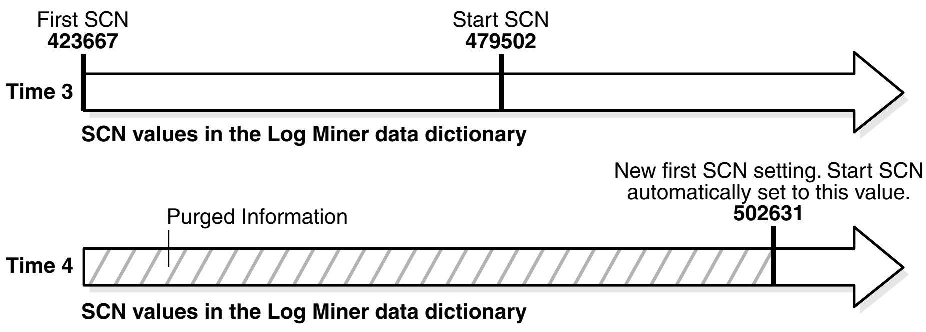
Figure 2–4 shows how Oracle automatically purges LogMiner data dictionary information prior to a new first SCN setting, and how the start SCN is not changed if it is higher than the new first SCN setting.

Figure 2–4 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 2–5 shows how Oracle resets the start SCN automatically if it is lower than a new first SCN setting.

Figure 2–5 Start SCN Lower Than Reset First SCN


First SCN
423667
Start SCN
479502
Time 3
SCN values in the Log Miner data dictionary
New first SCN setting. Start SCN automatically set to this value.
502631
Purged Information
Time 4
SCN values in the Log Miner data dictionary

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

See Also:

- "First SCN and Start SCN" on page 2-24
- "Setting the Start SCN for an Existing Capture Process" on page 9-36
- The `DBMS_CAPTURE_ADM.ALTER_CAPTURE` procedure in the *PL/SQL Packages and Types Reference* for information about altering a capture process

The Streams Data Dictionary

Propagations and apply processes 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 propagations 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 may then propagate these internal messages to destination queues 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 3-23
- "Streams Data Dictionary for an Apply Process" on page 4-18

ARCHIVELOG Mode and a Capture Process

A local capture process reads online redo logs whenever possible and archived redo log files otherwise. A downstream capture process always reads archived redo log files from its source database. For this reason, the source database must be running in `ARCHIVELOG` mode when a capture process is configured to capture changes. You must keep an archived redo log file available until you are certain that no capture process will need that file.

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.

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.

See Also:

- *Oracle Database Administrator's Guide* for information about running a database in `ARCHIVELOG` mode
- "Displaying the Redo Log Files That Will Never Be Needed by Any Capture Process" on page 14-13

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 can be used to specify the amount of time a capture process runs before it is shut down automatically.

See Also:

- "Setting a Capture Process Parameter" on page 9-32
- This section does not discuss all of the available capture process parameters. See the `DBMS_CAPTURE_ADM.SET_PARAMETER` procedure in the *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 servers 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 2-29 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 events 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 (`c`*`nnn`*) remains the same.

Capture Process Rule Evaluation

A capture process evaluates changes it finds in the redo log against its positive and negative rule sets. The capture process evaluates a change against the negative rule set first. If one or more rules 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 the object level and schema 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 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.
   - 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 may not have enough information to determine whether the change will definitely satisfy both of its rule sets. In this case, the change is converted to an LCR for further evaluation.

- 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 may not have enough information to determine whether the change will definitely satisfy both of its rule sets. In this case, the change is converted to an LCR for further evaluation.
- 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 may not have enough information to determine whether the change will definitely satisfy both of its rule sets. In this case, the change is converted to an LCR for further evaluation.
- 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 may 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 LCRs 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:

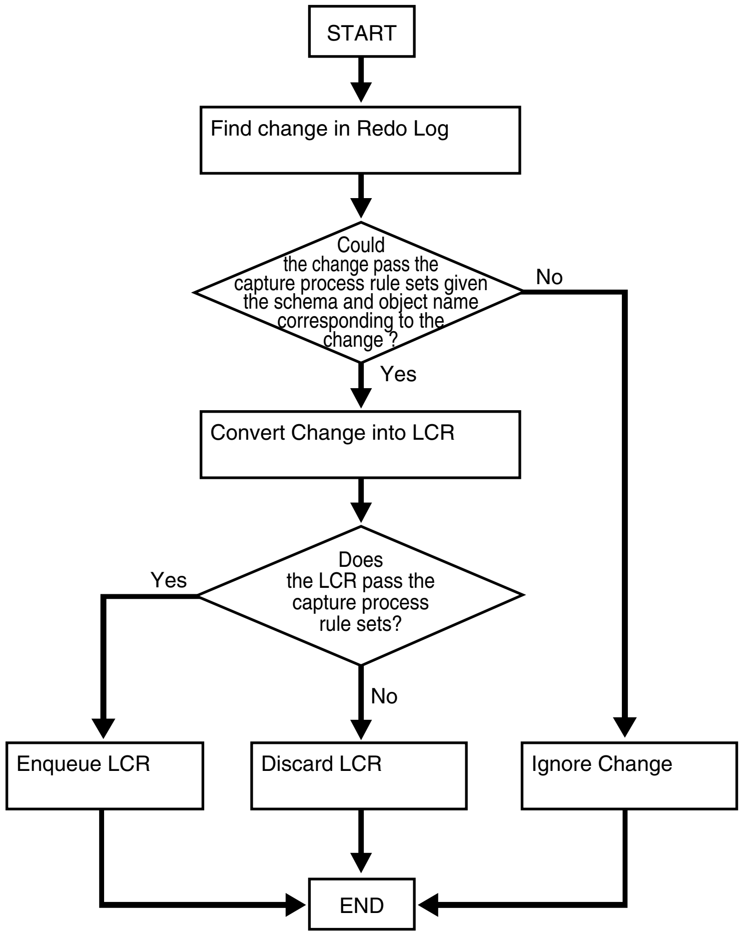
1. Finds the next change resulting from the `UPDATE` statement in the redo log.
2. Determines that the change resulted from an `UPDATE` statement to the `hr.employees` table and must be captured. If the change was made to a different table, then the capture process ignores the change.
3. Captures the 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 2-29
- Chapter 6, "How Rules Are Used In Streams" for more information about rule sets for Streams clients and for information about how events satisfy rule sets

Figure 2–6 illustrates capture process rule evaluation in a flowchart.

Figure 2–6 Flowchart Showing Capture Process Rule Evaluation


START
Find change in Redo Log
Could the change pass the capture process rule sets given the schema and object name corresponding to the change ?
No
Yes
Convert Change into LCR
Does the LCR pass the capture process rule sets?
Yes
No
Enqueue LCR
Discard LCR
Ignore Change
END

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 events in a queue and propagating events from one queue to another.

This chapter contains these topics:

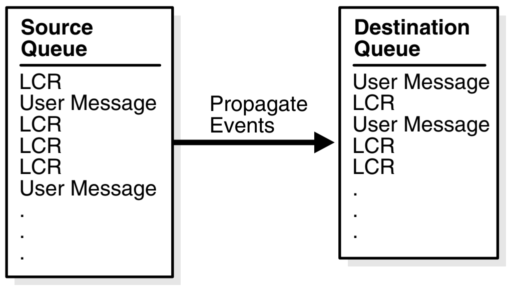
See Also: Chapter 10, "Managing Staging and Propagation"

Introduction to Event Staging and Propagation

Streams uses queues of type `SYS.AnyData` to stage events. In Streams, there are two types of events that can be encapsulated into a `SYS.AnyData` event and staged in a `SYS.AnyData` queue: logical change records (LCRs) and user messages. An LCR is an object that contains information about a change to a database object, while a user message is a message of a user-defined type created by users or applications. Both types of events can be used for information sharing within a single database or between databases.

Staged events can be consumed or propagated, or both. Staged events can be consumed by an apply process, by a messaging client, or by a user application. A running apply process implicitly dequeues events, but messaging clients and user applications explicitly dequeue events. Even after an event is consumed, it may remain in the queue if you also have configured Streams to propagate the event to one or more other queues or if message retention is specified for user-enqueued messages. The queues to which events are propagated may reside in the same database or in different databases than the queue from which the events are propagated. In either case, the queue from which the events are propagated is called the **source queue**, and the queue that receives the events 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 rules that control which events are propagated. The user who owns the source queue is the user who propagates events, and this user must have the necessary privileges to propagate events.

These privileges include the following:

- Execute privilege on the rule sets used by the propagation
- Execute privilege on all 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 events to a destination queue in a remote database, then the owner of the source queue must be able to use the propagation's database link, 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 used with Streams propagations.
- Message retention does not apply to LCRs captured by a capture process.

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 Events

Events can be enqueued in two ways:

- A capture process enqueues captured changes in the form of events containing LCRs. An event containing an LCR that was originally captured and enqueued by a capture process is called a **captured event**.
- A user application enqueues user messages encapsulated in events of type `SYS.AnyData`. These user messages can contain LCRs or any other type of message. Any user message that was explicitly enqueued by a user or an application is called a **user-enqueued event**. Events that were enqueued by a user procedure called from an apply process are also user-enqueued events.

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

Events can be dequeued in two ways:

- An apply process dequeues either captured or user-enqueued events. If the event contains an LCR, then the apply process either can apply it directly or call a user-specified procedure for processing. If the event 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 LCRs 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 events.
- A user application explicitly dequeues user-enqueued events and processes them. The user application may or may not use a Streams messaging client. Captured events cannot be dequeued by a user application. Captured events must be dequeued by an apply process. However, if a user procedure called by an apply process explicitly enqueues an event, then the event is a user-enqueued event and can be explicitly dequeued, even if the event was originally a captured event.

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

See Also:

- Chapter 2, "Streams Capture Process" for more information about the capture process
- "Messaging Clients" on page 3-11
- 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 events into a queue
- *Oracle Streams Replication Administrator's Guide* for more information about managing LCRs

Event Propagation Between Queues

You can use Streams to configure event propagation between two queues, which may reside in different databases. Streams uses job queues to propagate events.

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

A propagation may propagate all of the events in a source queue to a destination queue, or a propagation may propagate only a subset of the events. Also, a single propagation can propagate both captured and user-enqueued events. You can use rules to control which events in the source queue are propagated to the destination queue and which events 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 tags to avoid such a change cycling loop.

See Also:

- "Managing Streams Propagations and Propagation Jobs" on page 10-7
- *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 events based on rules that you define. For LCR events, 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 for the propagation or a negative rule set for the propagation.

If a rule evaluates to TRUE for an event, 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 an event, 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 LCR events 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 events, 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"

Ensured Event Delivery

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

- The event is processed by all relevant apply processes associated with the destination queue.
- The event is propagated successfully from the destination queue to all of its relevant destination queues.

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

This confirmation system ensures that events 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 may use more disk space.

There are two common reasons for source-queue growth:

- If an event cannot be propagated to a specified destination queue for some reason (such as a network problem), then the event 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 events to multiple destination queues, and one or more destination databases acknowledge successful propagation of events much more slowly than the other queues. In this case, the source queue can grow because the slower destination databases create a backlog of events that have already been acknowledged by the faster destination databases. In an environment such as this, consider creating more than one capture process to capture changes at the source database. In this case, you can use one source queue for the slower destination databases and another source queue for the faster destination databases.

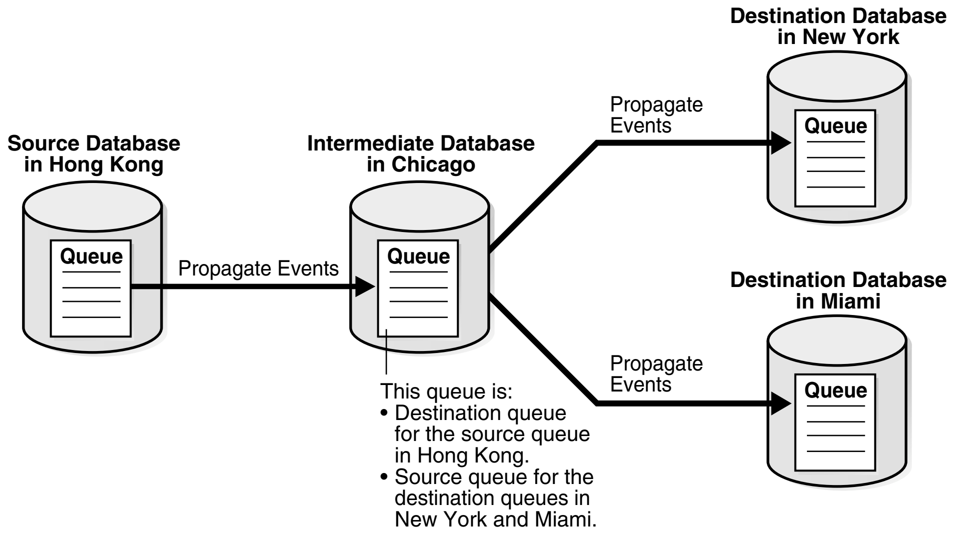
See Also:

- Chapter 2, "Streams Capture Process"
- "Monitoring a SYS.AnyData Queue and Messaging" on page 14-22

Directed Networks

A directed network is one in which propagated events may pass through one or more intermediate databases before arriving at a destination database. An event may or may not be processed by an apply process at an intermediate database. Using Streams, you can choose which events are propagated to each destination database, and you can specify the route that events 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


Source Database in Hong Kong
Queue
Propagate Events
Intermediate Database in Chicago
Queue
This queue is:
• Destination queue for the source queue in Hong Kong.
• Source queue for the destination queues in New York and Miami.
Propagate Events
Destination Database in New York
Queue
Propagate Events
Destination Database in Miami
Queue

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 events 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 events 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 may need to reconfigure the network and the Streams environment.

Queue Forwarding and Apply Forwarding

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

Apply forwarding means that the events being forwarded at an intermediate database are first processed by an apply process. These events 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 events because the events 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, an event is propagated through the directed network without being changed, assuming there are no capture or propagation transformations. With apply forwarding, events are applied and recaptured at intermediate databases and may be changed by conflict resolution, apply handlers, or apply transformations.
- With queue forwarding, a destination database must have a separate apply process to apply events from each source database. With apply forwarding, fewer apply processes may be required at a destination database because recapturing of events at intermediate databases may 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 events are recaptured at intermediate databases, the source database for an event can be the same as the intermediate database connected directly with the destination database.

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

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

- Performance may be improved because an event is captured only once.
- Less time may be required to propagate an event from the database where the event originated to the destination database, because the events are not applied and recaptured at one or more intermediate databases. In other words, latency may be lower with queue forwarding.
- The origin of an event can be determined easily by running the `GET_SOURCE_DATABASE_NAME` member procedure on the LCR contained in the event. If you use apply forwarding, then determining the origin of an event requires the use of Streams tags and apply handlers.

- Parallel apply may 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 may be required to reconfigure end-to-end capture, propagation, and apply of events, 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 to queue forwarding:

- A Streams environment may 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 may be easier to monitor and manage because fewer apply processes may 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 databases. The message payload can be a `BFILE` wrapped in a `SYS.AnyData` payload, or the message payload can be one or more `BFILE` attributes of an object wrapped in a `SYS.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 a `SYS.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 *PL/SQL Packages and Types Reference* for more information about transferring files with the `DBMS_FILE_TRANSFER` package

Messaging Clients

A messaging client consumes user-enqueued events when it is invoked by an application or a user. You use rules to specify which user-enqueued events in the queue are dequeued by a messaging client. These user-enqueued events may 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:

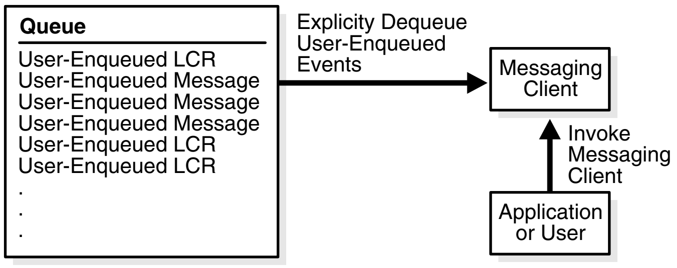
- `ADD_MESSAGE_RULE`
- `ADD_TABLE_RULES`
- `ADD_SUBSET_RULES`
- `ADD_SCHEMA_RULES`
- `ADD_GLOBAL_RULES`

When you create a messaging client, you specify the name of the messaging client and the SYS.AnyData queue from which the messaging client dequeues messages. These procedures also can add rules to the positive 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 sets. A messaging client can be associated with only one user, but one user may be associated with many messaging clients.

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

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 10-25

SYS.AnyData Queues and User Messages

Streams enables messaging with queues of type SYS.AnyData. These queues can stage user messages whose payloads are of SYS.AnyData type. A SYS.AnyData payload can be a wrapper for payloads of different datatypes.

By using SYS.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 a `SYS.AnyData` payload. To do this, you use the `Convert`*`data_type`* static functions of the `SYS.AnyData` type, where *`data_type`* is the type of object to wrap. These functions take the object as input and return a `SYS.AnyData` object.

You cannot enqueue `SYS.AnyData` payloads that contain payloads of the following datatypes into a `SYS.AnyData` queue:

- `CLOB`
- `NCLOB`
- `BLOB`

In addition, you cannot enqueue `SYS.AnyData` payloads that contain payloads of object types with attributes of these datatypes into a `SYS.AnyData` queue, nor object types that use type evolution or type inheritance.

Note:

- Payloads of `ROWID` datatype cannot be wrapped in a `SYS.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 10-20
- "Wrapping User Message Payloads in a SYS.AnyData Wrapper and Enqueuing Them" on page 10-20
- *Oracle Streams Advanced Queuing User's Guide and Reference* for more information relating to `SYS.AnyData` queues, such as wrapping payloads in a `SYS.AnyData` wrapper, programmatic environments for enqueuing messages into and dequeuing messages from a `SYS.AnyData` queue, propagation, and user-defined types
- *PL/SQL Packages and Types Reference* for more information about the `SYS.AnyData` type

SYS.AnyData Queues and Oracle Real Application Clusters

You can configure a `SYS.AnyData` queue to stage and propagate captured and user-enqueued events in a Real Application Clusters (RAC) environment. In a RAC environment, only the owner instance may have a buffer for a queue, but different instances may have buffers for different queues. A buffered queue is System Global Area (SGA) memory associated with a `SYS.AnyData` queue that contains only captured events. Buffered queues are discussed in more detail later in this chapter.

A `SYS.AnyData` queue that contains only user-enqueued events behaves the same as a typed queue in a RAC environment. However, if a `SYS.AnyData` queue contains or will contain captured events in a RAC environment, then each propagation that propagates captured events to a RAC destination database must use an instance-specific database link that refers to the owner instance of the destination queue. If the propagation connects to any other instance, then the propagation will raise an error.

Each capture process and apply process is started on the owner instance for its `SYS.AnyData` queue, even if the start procedure is run on a different instance. 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. If this happens, then database links from remote source queues must be reconfigured manually to connect to the instance that owns the destination queue.

Streams processes and jobs support primary instance and secondary instance specifications for queue tables. 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.

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

See Also:

- "Streams Capture Processes and Oracle Real Application Clusters" on page 2-27
- "Streams Apply Processes and Oracle Real Application Clusters" on page 4-13
- "Buffered Queues" on page 3-16
- *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
- *PL/SQL Packages and Types Reference* for more information about the `ALTER_QUEUE_TABLE` procedure

Streams Staging and Propagation Architecture

In general, `SYS.AnyData` queues use the same infrastructure as typed queues. However, unlike typed queues, which stage all events in a queue table, `SYS.AnyData` queues have a buffered queue to stage captured events in shared memory. This section describes buffered queues, propagation jobs, and secure queues, and how they are used in Streams. In addition, this section discusses how transactional queues handle captured and user-enqueued events, as well as the need for a Streams data dictionary at databases that propagate captured events.

This section contains the following topics:

- 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
- *PL/SQL Packages and Types Reference* for more information about the `DBMS_JOB` package

Buffered Queues

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

- System Global Area (SGA) memory associated with a `SYS.AnyData` queue that contains only captured events
- Part of a queue table for a `SYS.AnyData` queue that stores captured events that have spilled from memory

A buffered queue enables Oracle to optimize captured events by buffering captured events in the SGA instead of always storing them in a queue table. This buffering of captured events happens in any database where captured events are staged in a `SYS.AnyData` queue. Such a database may be a source database, an intermediate database, or a destination database. Captured events are always stored in a buffered queue, but user-enqueued LCR events and user-enqueued non-LCR events are always stored in queue tables, not in buffered queues. Also, when a transaction is moved to the error queue, all events in the transaction are stored in a queue table, not in a buffered queue.

Buffered queues improve performance, but some of the information in a buffered queue may 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.

In a single database, you can specify that Streams memory be allocated from a new pool in the SGA called the Streams pool. To configure the Streams pool, specify the size of the pool in bytes using the `STREAMS_POOL_SIZE` initialization parameter. If the size of the Streams pool is greater than zero, then any SGA memory used by Streams is allocated from the Streams pool. If the size of the Streams pool is zero, then the memory used by Streams is allocated from the shared pool and may use up to 10% of the shared pool.

Captured events in a buffered queue may 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 captured events. Captured events 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.

See Also:

- "Setting Initialization Parameters Relevant to Streams" on page 8-6 for more information about the `STREAMS_POOL_SIZE` initialization parameter
- *Oracle Database Concepts* for more information about the SGA
- *Oracle Streams Replication Administrator's Guide* for information about performing database point-in-time recovery on a database in a Streams environment

Propagation Jobs

A Streams propagation is configured internally using the `DBMS_JOBS` package. Therefore, a propagation job is the mechanism that propagates events from a source queue to a destination queue. Like other jobs configured using the `DBMS_JOBS` package, propagation jobs have an owner, and they use job queue processes (`J`*`nnn`*) as needed to execute jobs.

A propagation job may be used by more than one propagation. All destination queues at a database receive events from a single source queue through a single propagation job. By using a single propagation job for multiple destination queues, Streams ensures that an event 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:

- A single propagation job propagates all events that use a particular database link, even if the database link is used by more than one propagation to propagate events to multiple destination queues.
- The source queue owner performs the propagation, but the propagation job is owned by the user who creates it. These two users may or may not be the same.

Propagation Scheduling and Streams Propagations

A propagation schedule specifies how often a propagation job propagates events from a source queue to a destination queue. Therefore, all propagations that use a propagation job have the same propagation schedule. A default propagation schedule is established for the new propagation job when you create the propagation job using one of the following procedures:

- 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

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.

If you want to alter the default schedule for a propagation job, then use the `ALTER_PROPAGATION_SCHEDULE` procedure in the `DBMS_AQADM` package.

> **See Also:** "Altering the Schedule of a Propagation Job" on page 10-12

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 events. 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` with the `ENABLE RESTRICTED SESSION` clause, 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 may eventually come to a halt.

Secure Queues

Secure queues are queues 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 clients enqueue events into a queue and dequeue events from a queue.

Secure Queues and the SET_UP_QUEUE Procedure

All SYS.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 events into and dequeue events 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, an event cannot be enqueued into a queue unless a subscriber who can dequeue the event 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 will skip queue creation, but it will configure the user specified by queue_user as a secure queue user of the queue.

If you create a SYS.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 may 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 may 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` may 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 may also want to drop the agent if it is no longer needed.

Note: No configuration is necessary for propagations 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 10-3
- "Disabling a User from Performing Operations on a Secure Queue" on page 10-5
- *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 one in which user-enqueued events 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 events 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 event is its own transaction. That is, an apply process performs a `COMMIT` after each user-enqueued event it applies. In either case, the user-enqueued events may or may not contain user-created LCRs.

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

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

Event Type	Transactional Queue	Nontransactional Queue
Captured Events	Apply process preserves the original transaction	Apply process preserves the original transaction
User-Enqueued Events	Apply a user-specified group of user-enqueued events as one transaction	Apply each user-enqueued event in its own transaction

See Also:

- "Managing SYS.AnyData Queues" on page 10-2
- *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 event as small as possible because the event can store numbers rather than names internally.

The mapping information in the Streams data dictionary at the source database is needed to evaluate rules at any database that propagates the captured events 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 events from the source database.

The Streams data dictionary information contained in these internal messages in a queue may or may not be propagated by a propagation. Which Streams data dictionary information to propagate depends on the rule sets 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 may 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 2-45
- 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:

See Also: Chapter 11, "Managing an Apply Process"

Introduction to the Apply Process

An apply process is an optional Oracle background process that dequeues logical change records (LCRs) and user messages from a specific queue and either applies each one directly or passes it as a parameter to a user-defined procedure. 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 `SYS.AnyData` and can contain any user message, including a user-created LCR.

Note: An apply process can be associated only with a `SYS.AnyData` queue, not with a typed queue.

Apply Process Rules

An apply process applies changes based on rules 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 for the apply process or a 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 LCR events 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 events, you can create your own rules to control apply process behavior.

See Also:

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

Event Processing with an Apply Process

An apply process is a flexible mechanism for processing the events in a queue. You have options to consider when you configure one or more apply processes for your environment. This section discusses the types of events that an apply process can apply and the ways that it can apply them.

Processing Captured and User-Enqueued Events with an Apply Process

A single apply process can apply either captured events or user-enqueued events, but not both. If a queue at a destination database contains both captured and user-enqueued events, then the destination database must have at least two apply processes to process the events.

When you create an apply process using one of the following procedures in the `DBMS_STREAMS_ADM` package, the apply process applies only captured events:

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

When you create an apply process using the `ADD_MESSAGE_RULE` procedure in the `DBMS_STREAMS_ADM` package, the apply process applies only user-enqueued events.

When you create an apply process using the `CREATE_APPLY` procedure in the `DBMS_APPLY_ADM` package, you use the `apply_captured` parameter to specify whether the apply process applies captured or user-enqueued events. By default, the `apply_captured` parameter is set to `false` for an apply process created with this procedure. Therefore, by default, an apply process created with the `CREATE_APPLY` procedure in the `DBMS_APPLY_ADM` package applies user-enqueued events.

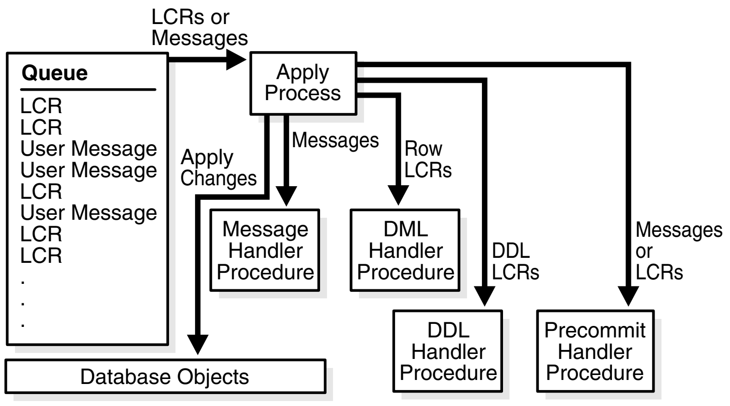
See Also:

- "Introduction to Event Staging and Propagation" on page 3-2 for more information about captured and user-enqueued events
- *PL/SQL Packages and Types Reference* for more information about the `CREATE_APPLY` procedure

Event Processing Options with an Apply Process

Your options for event processing depend on whether or not the event received by an apply process is an LCR event. Figure 4–1 shows the event processing options for an apply process.

Figure 4–1 The Apply Process



LCR Event Processing

An apply process either can apply captured LCRs or user-enqueued LCRs, but not both. Regarding captured LCRs, an apply process can apply captured LCRs 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 LCR, the source database is the database where the change encapsulated in the LCR was generated in the redo log.

Captured LCRs from multiple databases may be sent to a single destination queue. However, if a single queue contains captured LCRs 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 LCRs from exactly one source database using rules. Regarding user-enqueued events containing LCRs (not captured events), a single apply process can apply these user-enqueued events, even if they are from multiple source databases.

Also, each apply process can apply captured LCRs from only one capture process. If there are multiple capture processes 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 Corporation recommends that each `SYS.AnyData` queue used by a capture process or apply process have captured LCRs 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.

You can configure an apply process to process a captured or user-enqueued event that contains an LCR in the following ways: directly apply the LCR event or pass the LCR event as a parameter to a user procedure for processing. The following sections explain these options.

Apply the LCR Event Directly If you use this option, then an apply process applies the LCR event 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 may 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 Event If you use this option, then an apply process passes the LCR event as a parameter to a user procedure for processing. The user procedure can process the LCR event in a customized way.

A user procedure that processes row LCRs resulting from DML statements is called a **DML handler**, while 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 may have one DML handler to process `INSERT` operations and a different DML handler to process `UPDATE` operations.

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 save point 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 either may set a DML handler for a specific apply process, or you may 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 with 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 may be used in non-replication environments as well. For example, such handlers may be used to record changes made to database objects without replicating these changes.

Attention: Do not modify `LONG`, `LONG RAW` or LOB column data in an LCR. This includes DML handlers, error handlers, and rule-based transformation functions.

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
- *PL/SQL Packages and Types Reference* for more information about the `EXECUTE` member procedure for LCR types
- "Rule-Based Transformations" on page 6-63
- *Oracle Streams Replication Administrator's Guide* for more information about DML handlers and DDL handlers

Non-LCR User Message Processing

A user-enqueued event 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 non-LCR user messages 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 may format a user message into an electronic mail message. In this case, the user message may contain the attributes you would expect in an electronic mail message, such as `from`, `to`, `subject`, `text_of_message`, and so on. A message handler could convert these user messages into electronic mail messages and send them 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 events in the queue. If parallelism is greater than `1` for an apply process that applies user-enqueued messages, then these messages may be dequeued by a message handler in any order. Therefore, if dependencies exist between these messages in your environment, then Oracle Corporation recommends that you set apply process parallelism to `1`.

See Also: "Managing the Message Handler for an Apply Process" on page 11-17

Audit Commit Information for Events Using Precommit Handlers

You can use a precommit handler to audit commit directives for captured events and transaction boundaries for user-enqueued events. 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 may work with a DML handler or a message handler.

For example, a handler may improve performance by caching data for the length of a transaction. This data may include cursors, temporary LOBs, data from an event, 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 Events 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 may be propagated to destination queues, 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 Events A user or application may 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 may 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, nor 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 11-18

Summary of Event Processing Options

Table 4–1 summarizes the event processing options available when you are using one or more of the apply handlers described in the previous sections. Apply handlers are optional for row LCRs and DDL LCRs because an apply process can apply these events directly. However, a message handler is required for processing non-LCR user messages. In addition, an apply process dequeues an event only if the event satisfies the rule sets for the apply process. In general, an event satisfies the rule sets for an apply process if *no rules* in the negative rule set evaluate to `TRUE` for the event, and *at least one rule* in the positive rule set evaluates to `TRUE` for the event.

Table 4–1 Summary of Event Processing Options

Apply Handler	Type of Event	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	Non-LCR User Message	Create error transaction (if no message handler exists)	Entire apply process
Precommit Handler	Commit directive for transactions that include row LCRs or non-LCR user messages	Commit transaction	Entire apply process

In addition to the event 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 events into a specified destination queue. Also, you can control event 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 clients and for information about how events satisfy rule sets
- "Specifying Event Enqueues by Apply Processes" on page 11-21
- "Specifying Execute Directives for Apply Processes" on page 11-23

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 may 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:

- *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

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`

The 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, `REF`s, varrays, nested tables, and Oracle-supplied types). The 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 2-8
- *Oracle Database SQL Reference* for more information about these datatypes

Streams Apply Processes and RESTRICTED SESSION

When the restricted session is enabled during system startup by issuing a `STARTUP RESTRICT` statement, apply processes 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 the restricted session is enabled in a running database by the SQL statement `ALTER SYSTEM` with the `ENABLE RESTRICTED SESSION` clause, it does not affect any running apply processes. These apply processes continue to run and apply events. If a stopped apply process is started in a restricted session, then the apply process does not 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 a Real Application Clusters (RAC) environment. Each apply process is started on the owner instance for its `SYS.AnyData` queue, even if the start 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 server processes run on a single instance.

See Also:

- "SYS.AnyData Queues and Oracle Real Application Clusters" on page 3-14 for information about primary and secondary instance ownership for queues
- "Streams Capture Processes and Oracle Real Application Clusters" on page 2-27
- *Oracle Database Reference* for more information about the `DBA_QUEUE_TABLES` data dictionary view
- "Persistent Apply Process Status Upon Database Restart" on page 4-22

Apply Process Architecture

You can create, alter, start, stop, and drop an apply process, and you can define apply process rules that control which events an apply process dequeues from its queue. Events applied by an apply process are applied by an **apply user**. The apply user applies all changes that satisfy the apply process rule sets. In addition, the apply user runs all rule-based transformations specified by the rules in these rule sets. The apply user also runs user-defined apply handlers.

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 rule-based transformation functions specified for rules in the positive rule set, execute privilege on any apply handlers, and privileges to dequeue events from the apply process queue. An apply process can be associated with only one user, but one user may be associated with many apply processes.

See Also: "Configuring a Streams Administrator" on page 8-2 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 events. The reader server is a parallel execution server that computes dependencies between LCRs and assembles events 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 handlers. For non-LCR messages, the apply servers pass the events to the message handler. 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 events, 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 with 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.

For example, consider these two transactions:

1. A row is inserted into a table.
2. The same row is updated to change certain column values.

In this case, transaction 2 is dependent on transaction 1, because the row cannot be updated until after it is inserted into the table. Suppose these transactions are captured from the redo log at a source database, propagated to a destination database, and applied at the destination database. Apply server A handles the insert transaction, and apply server B handles the update transaction.

If apply server B is ready to apply the update transaction before apply server A has applied the insert transaction, then apply server B waits for instructions from the coordinator process. After apply server A has applied the insert transaction, the coordinator process instructs apply server B to apply the update transaction.

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:

- `IDLE` - Performing no work
- `DEQUEUE MESSAGES` - Dequeuing events from the apply process queue

- `SCHEDULE MESSAGES` - Computing dependencies between events and assembling events into transactions

> **See Also:** "Displaying Information About the Reader Server for Each Apply Process" on page 14-38 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 14-40 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:

- `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 14-44 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 rules may 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_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 events only at the local database. To create an apply process that applies events 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: *PL/SQL Packages and Types Reference* for more information about the following procedures, which can be used 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_CAPTURE_ADM.CREATE_APPLY`

Also, see *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 event as small as possible because a captured event can often use numbers rather than names internally.

Unless a captured event is passed as a parameter to a 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 event. 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 events from the source database.

See Also:

- "The Streams Data Dictionary" on page 2-45
- "Streams Data Dictionary for Propagations" on page 3-23

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 can be used to specify 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 11-16
- This section does not discuss all of the available apply process parameters. See the `DBMS_APPLY_ADM.SET_PARAMETER` procedure in the *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 may concurrently apply transactions. For example, if `parallelism` is set to `5`, then an apply process uses a total of five apply servers. In addition, 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, which may 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 may 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 4-14 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 may apply transactions at the destination database in an order that is different from the commit order at the source database. Only nondependent transactions can be applied in a different order than 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 commits applied transactions in the order in which they were committed at the source database. This setting is the default.
- `none`: An apply process may commit non-dependent transactions in any order. Performance is best if you specify this value.

If you specify `none`, then it is possible that a destination database may commit 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 may 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 either can instruct an apply process to become disabled when it encounters an error, or you can allow the apply process to continue applying transactions after it encounters an error.

See Also: "The Error Queue" on page 4-22

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 processes 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.

The error queue stores information about transactions that could not be applied successfully by the apply processes running in a database. A transaction may include many events, and when an unhandled error occurs during apply, an apply process automatically moves all of the events in the transaction that satisfy the apply process rule sets to the error queue.

You can correct the condition that caused an error and then reexecute the error transaction. 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 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 handlers 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 may modify the row LCR in some way to avoid a repeat of the same error. The row LCR is passed to the DML handler when you reexecute the error containing the row LCR using the `EXECUTE_ERROR` or `EXECUTE_ALL_ERRORS` procedure in the `DBMS_APPLY_ADM` package.

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 queues in the database. When you create a `SYS.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 may use a single queue table, and each queue table has one exception queue. Therefore, a single exception queue may 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 11-32
- "Checking for Apply Errors" on page 14-48
- "Displaying Detailed Information About Apply Errors" on page 14-50
- "Managing an Error Handler" on page 11-25
- Chapter 6, "How Rules Are Used In Streams" for more information about rule sets for Streams clients and for information about how events satisfy rule sets
- *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 12, "Managing Rules and Rule-Based Transformations"
- Chapter 17, "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. Rules 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.

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.

Note: A rule must be in a rule set for it to be evaluated.

Rule Condition

A **rule condition** combines one or more expressions 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 may 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 may 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 enable you to refer to data that is not stored in a table. A variable may also improve performance by replacing a commonly occurring expression. Performance may improve because, instead of evaluating the same expression multiple times, the variable is evaluated once.

A rule condition may also contain an evaluation of a call to a subprogram. These conditions are evaluated in the same way as other conditions. That is, they evaluate 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 either of the following forms:

- *`simple_rule_expression condition constant`*
- *`constant condition simple_rule_expression`*

In a simple rule condition, a *`simple_rule_expression`* is one of the following:

- Table column
- Variable
- Variable attribute
- Method result where the method takes no arguments and the method result can be returned by the variable method function, so that the expression is either a numerical or character type

For table columns, variables, and variable attributes, all numeric (`NUMBER`, `FLOAT`, `DOUBLE`, `INTEGER`) and character (`CHAR`, `VARCHAR2`) types are supported. Use of other types of expressions results in non-simple rule conditions.

In a simple rule condition, a *`condition`* is one of the following:

- `<=`
- `<`
- `=`
- `>`
- `>=`
- `!=`

- `IS NULL`
- `IS NOT NULL`

Use of other conditions results in non-simple rule conditions.

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"`

Therefore, the following conditions are simple rule conditions:

- `tab1.col = 5`
- `tab2.col != 5`
- `:v1 > 'aaa'`
- `:v2.a1 < 10.01`
- `:v3.m() = 10`
- `:v4 IS NOT NULL`

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's condition causes the rule to be non-simple.

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 `DBMS_RULE.EVALUATE` 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

Rule Evaluation Context

A **rule evaluation context** is a database object that defines external data that can be referenced in rule conditions. The external data can exist as variables, table data, or both. The following analogy may be helpful: If the rule condition were the `WHERE` clause in a SQL query, then the external data in the rule's evaluation context would be the tables and bind variables referenced in the `FROM` clause of the query. That is, the expressions 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.

Explicit and Implicit Variables

The value of a variable referenced in a rule condition may be explicitly specified when the rule is evaluated, or the value of a variable may 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 may reduce the complexity of the application using the rules engine. For example, a variable may call a function that returns a value based on the data being evaluated.
- The caller may 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 may improve security if the variable value contains confidential information.
- The variable may 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:

- *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

A single rule 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 may choose to 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 can associate the function with the 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 may 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 may 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: *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

A **rule 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 conditions 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 avoid naming conflicts by using different names or prefixes of names.
- Do not use $ and # in names to avoid conflicts with Oracle-supplied action context names.

You can add a name-value pair to an action context using the `ADD_PAIR` member procedure of the `RE$NV_LIST` type. You can remove a name-value pair from an action context using the `REMOVE_PAIR` member procedure of the `RE$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, nor object types that use type evolution or type inheritance.

Note: Streams uses action contexts for rule-based transformations and, when subset rules are specified, for internal transformations that may be required on LCRs containing `UPDATE` operations. Streams also uses action contexts to specify a destination queue into which an apply process enqueues events that satisfy the rule. In addition, Streams uses action contexts to specify whether or not an event that satisfies an apply process rule is executed by the apply process.

See Also:

- "Streams and Action Contexts" on page 6-50
- "Creating a Rule With an Action Context" on page 12-5 and "Altering a Rule" on page 12-7 for examples that add and modify name-value pairs
- *PL/SQL Packages and Types Reference* for more information about the `RE$NV_LIST` type

Rule Set Evaluation

The rules engine evaluates rule sets 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 may have more information than the information that the client sends to the rules engine.

The information specified by the client when it calls the DBMS_RULE.EVALUATE procedure includes the following:

- 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 variable's attributes. 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 directly used or interpreted by the rules engine. Instead, it is passed to client callbacks, such as an evaluation function, a variable value function (for implicit variables), and a variable method function.

The client also can send other information about how to evaluate an event against the rule set using the DBMS_RULE.EVALUATE procedure. For example, the caller may 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:

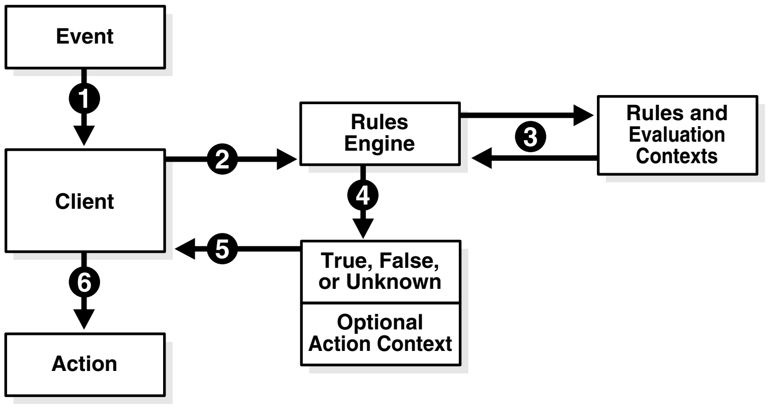
- One version of the procedure returns all of the rules that evaluate to `TRUE` in one list or all of the rules that evaluate to `MAYBE` in one list, and the two `OUT` parameters for this version of the procedure are `true_rules` and `maybe_rules`. That is, the `true_rules` parameter returns rules in one list that evaluate to `TRUE`, and the `maybe_rules` parameter returns rules in one list that may 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 may 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 may contain information or may be `NULL`.
6. The client performs actions based on the results returned by the rules engine. The rules engine does not perform actions based rule evaluations.

Figure 5–1 Rule Set Evaluation



See Also:

- *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 6-61 for information about Streams clients 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 may reference columns, variables, or attributes that are unavailable, while some other rules may 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 may or may 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.

For example, given the rules in the previous 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 5-4

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 sets 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 contexts 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 5-2 for more information about these database objects
- *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 rules are used in Streams.

This chapter contains these topics:

See Also:

- Chapter 5, "Rules" for more information about rules
- Chapter 12, "Managing Rules and Rule-Based Transformations"

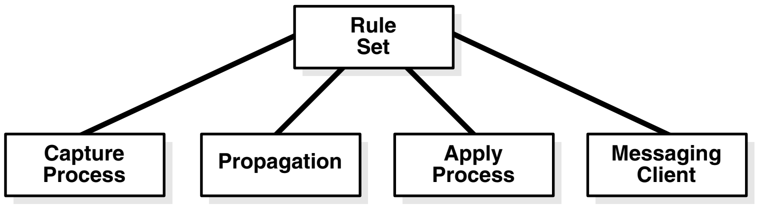
Overview of How Rules Are Used In Streams

In Streams, each of the following mechanisms is a client of a rules engine, when the mechanism is associated with one or more rule sets:

- 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 processes, propagations, apply processes, and messaging clients within the same database. Also, a single rule set may 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 an event satisfies its rule sets. In general, an event satisfies the rule sets for a Streams client if *no rules* in the negative rule set evaluate to `TRUE` for the event, and *at least one rule* in the positive rule set evaluates to `TRUE` for the event. "Rule Sets and Rule Evaluation of Events" on page 6-4 contains more detailed information about how an event 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 events that a propagation propagates from one queue to another or discards. That is, if an event in a queue satisfies the rule sets for a propagation, then the propagation propagates the event. If an event in a queue does not satisfy the rule sets for a propagation, then the propagation discards the event.
- Specify the events that an apply process retrieves from a queue or discards. That is, if an event in a queue satisfies the rule sets for an apply process, then the event is dequeued and processed by the apply process. If an event in a queue does not satisfy the rule sets for an apply process, then the apply process discards the event.
- Specify the user-enqueued events that a messaging client dequeues from a queue or discards. That is, if a user-enqueued event in a queue satisfies the rule sets for a messaging client, then the user or application that is using the messaging client dequeues the event. If a user-enqueued event 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 event.

In the case of a propagation or an apply process, the events evaluated against the rule sets can be captured events or user-enqueued events.

If there are conflicting rules 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 an event if either rule evaluates to `TRUE` for the event. 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"
- "Event Propagation Between Queues" on page 3-5
- Chapter 4, "Streams Apply Process"
- "Messaging Clients" on page 3-11

Rule Sets and Rule Evaluation of Events

Streams clients perform the following tasks based on rules:

- A capture process captures changes in the redo log, converts the changes into logical change record (LCR) events, and enqueues these events into the capture process queue.
- A propagation propagates either captured or user-enqueued events, or both, from a source queue to a destination queue.
- An apply process dequeues either captured or user-enqueued events from its queue and applies these events directly or sends the events to an apply handler.
- A messaging client dequeues user-enqueued events from its queue.

These Streams clients are all clients of the rules engine. A Streams client performs its task for an event when the event satisfies the rule sets used by the Streams client. A Streams client may 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 events 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 6-6

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 an event if any rule in the positive rule set evaluates to `TRUE` for the event. However, if all of the rules in a positive rule set evaluate to `FALSE` for the event, then the Streams client discards the event.

Streams Client With a Negative Rule Set Only

A Streams client with a negative rule set, but no positive rule set, discards an event if any rule in the negative rule set evaluates to `TRUE` for the event. However, if all of the rules in a negative rule set evaluate to `FALSE` for the event, then the Streams client performs its task for the event.

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 an event. If any rule in the negative rule set evaluates to `TRUE` for the event, then the event is discarded, and the event is never evaluated against the positive rule set.

However, if all of the rules in the negative rule set evaluate to `FALSE` for the event, then the event 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 an event if any rule in the positive rule set evaluates to `TRUE` for the event. If all of the rules in a positive rule set evaluate to `FALSE` for the event, then the Streams client discards the event.

Streams Client With One or More Empty Rule Sets

A Streams client may have one or more empty rule sets. 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 events.
- 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 rules, 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 events, 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 events
None	Exists with rules	Performs its task for events that evaluate to `TRUE` against the positive rule set
Exists with rules	None	Discards events that evaluate to `TRUE` against the negative rule set, and performs its task for all other events
Exists with rules	Exists with rules	Discards events that evaluate to `TRUE` against the negative rule set, and performs its task for remaining events 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 events
Exists but is empty	Exists with rules	Performs its task for events that evaluate to `TRUE` against the positive rule set
None	Exists but is empty	Discards all events
Exists but is empty	Exists but is empty	Discards all events
Exists with rules	Exists but is empty	Discards all events

System-Created Rules

A Streams client performs its task for an event if the event satisfies its rule sets. A system-created rule may 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.

Table 6–2 Types of Tasks and Rule Levels (Cont.)

Task	Table Rule	Schema Rule	Global Rule
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 may include conditions that modify the Streams client behavior beyond the descriptions in Table 6–2. For example, some rules may 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 conditions 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

Table 6–3 System-Created Rule Conditions Created by DBMS_STREAMS_ADM Package (Cont.)

Rule Condition Evaluates to TRUE for	Streams Client	Create Using Procedure
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 events 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 apply process's queue	Apply Process	`ADD_GLOBAL_RULES`
All DDL LCRs in the apply process's queue	Apply Process	`ADD_GLOBAL_RULES`
All row LCRs in the apply process's queue relating to the tables in a particular schema	Apply Process	`ADD_SCHEMA_RULES`
All DDL LCRs in the apply process's queue relating to a particular schema and any of the database objects in the schema	Apply Process	`ADD_SCHEMA_RULES`
All row LCRs in the apply process's queue relating to a particular table	Apply Process	`ADD_TABLE_RULES`
All DDL LCRs in the apply process's queue relating to a particular table	Apply Process	`ADD_TABLE_RULES`
All row LCRs in the apply process's queue relating to a subset of rows in a particular table	Apply Process	`ADD_SUBSET_RULES`
All user-enqueued events in the apply process's queue of the specified type that satisfy the user-specified rule condition	Apply Process	`ADD_MESSAGE_RULE`
All user-enqueued row LCRs in the messaging client's queue	Messaging Client	`ADD_GLOBAL_RULES`
All user-enqueued DDL LCRs in the messaging client's queue	Messaging Client	`ADD_GLOBAL_RULES`
All user-enqueued row LCRs in the messaging client's queue relating to the tables in a particular schema	Messaging Client	`ADD_SCHEMA_RULES`

Table 6–3 System-Created Rule Conditions Created by DBMS_STREAMS_ADM Package (Cont.)

Rule Condition Evaluates to TRUE for	Streams Client	Create Using Procedure
All user-enqueued DDL LCRs in the messaging client's queue 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 messaging client's queue relating to a particular table	Messaging Client	`ADD_TABLE_RULES`
All user-enqueued row LCRs in the messaging client's queue relating to a subset of rows in a particular table	Messaging Client	`ADD_SUBSET_RULES`
All user-enqueued events in the messaging client's queue 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 may 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 either adds the rules to the positive rule set, or it adds the rules 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$_EVALUATION_CONTEXT` evaluation context, which is an Oracle-supplied evaluation context for Streams environments. Also, global, schema, table, and subset rules use the `SYS.STREAMS$_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. Also, if the ADD_MESSAGE_RULE or the ADD_MESSAGE_PROPAGATION_RULE procedure creates a rule set, then the rule set does 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 events. They do not create rules for the results of DML changes or DDL changes to a table.

When you create propagation rules for captured events, Oracle Corporation recommends that you specify a source database for the changes. An apply process uses transaction control events to assemble captured events into committed transactions. These transaction control events, such as COMMIT and ROLLBACK, contain the name of the source database where the event occurred. To avoid unintended cycling of these events, 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 this section should be completed by a Streams administrator that has been granted the appropriate privileges, unless specified otherwise.
- Some prerequisites are required for the examples in this section to work. For example, a queue specified by a procedure parameter must exist.

See Also:

- "Rule Sets and Rule Evaluation of Events" on page 6-4 for information about how events satisfy the rule sets for a Streams client
- *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 6-45
- "Logical Change Records (LCRs)" on page 2-2
- "Complex Rule Conditions" on page 6-58

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 two rules for each type of change.

A single global rule in the positive rule set for a capture process means that the capture process captures either the results of 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 either the results of 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 6-56

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 rules 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 also may 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() in the previous example 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() in the previous example 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.

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 transformations for these changes upon capture, then you may 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 Events" on page 6-4 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 two rules for each type of change to the schema.

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 6-56

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 rules 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 may 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 two rules for each type of change for a specific table.

A single table rule in the positive rule set for a capture process means that the capture process captures either the results of 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 either the results of 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 may 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. You can create subset rules for capture processes, apply processes, and messaging clients using the `ADD_SUBSET_RULES` procedure, and you can create subset rules for propagations 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:

- Creating subset rules for tables that have one or more LOB, LONG, LONG RAW, or user-defined type columns is not supported.
- 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 higher.

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 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 6-27 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 may 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 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 6-27 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

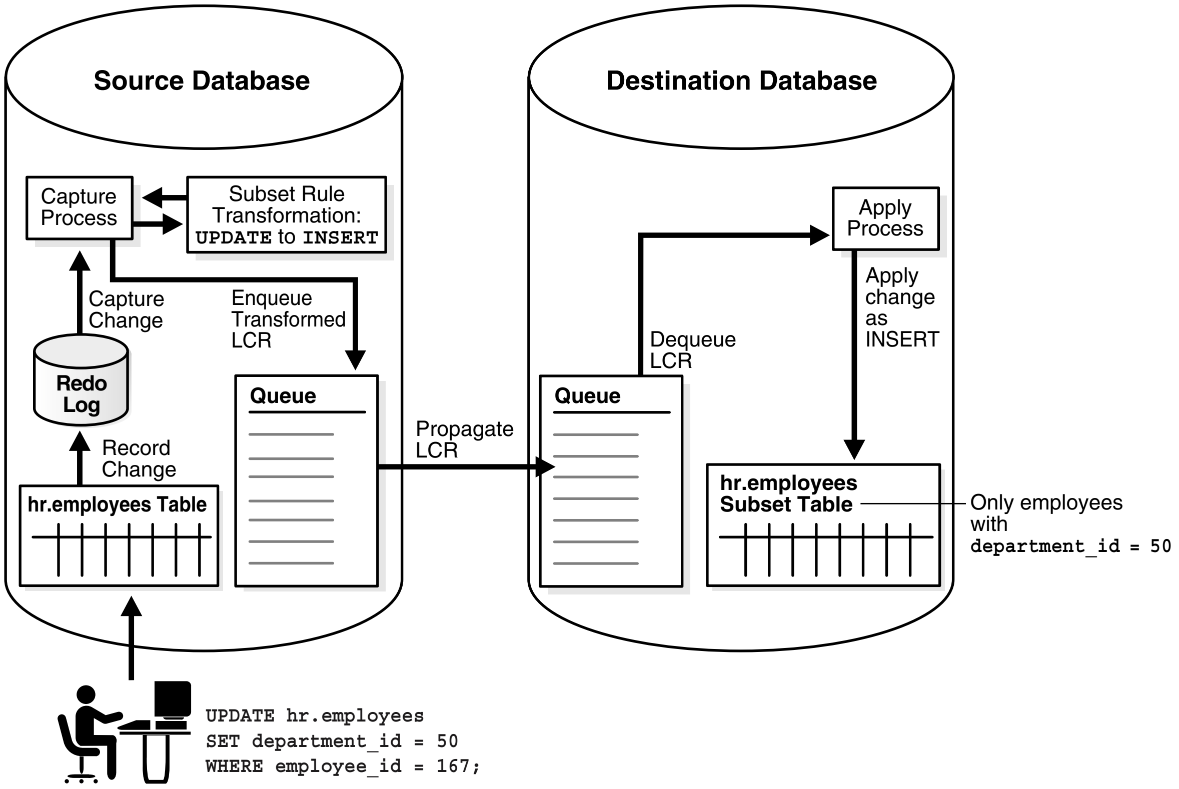
When you use subset rules, an update operation may be converted into an insert or delete operation when it is captured, propagated, applied, or dequeued. This automatic conversion is called **row migration** and is performed by an internal transformation specified automatically in a subset rule's action context. The following sections describe row migration during capture, propagation, apply, and dequeue.

Attention: Subset rules should only reside in positive rule sets. You should not add subset rules to negative rule sets. Doing so may 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 may 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

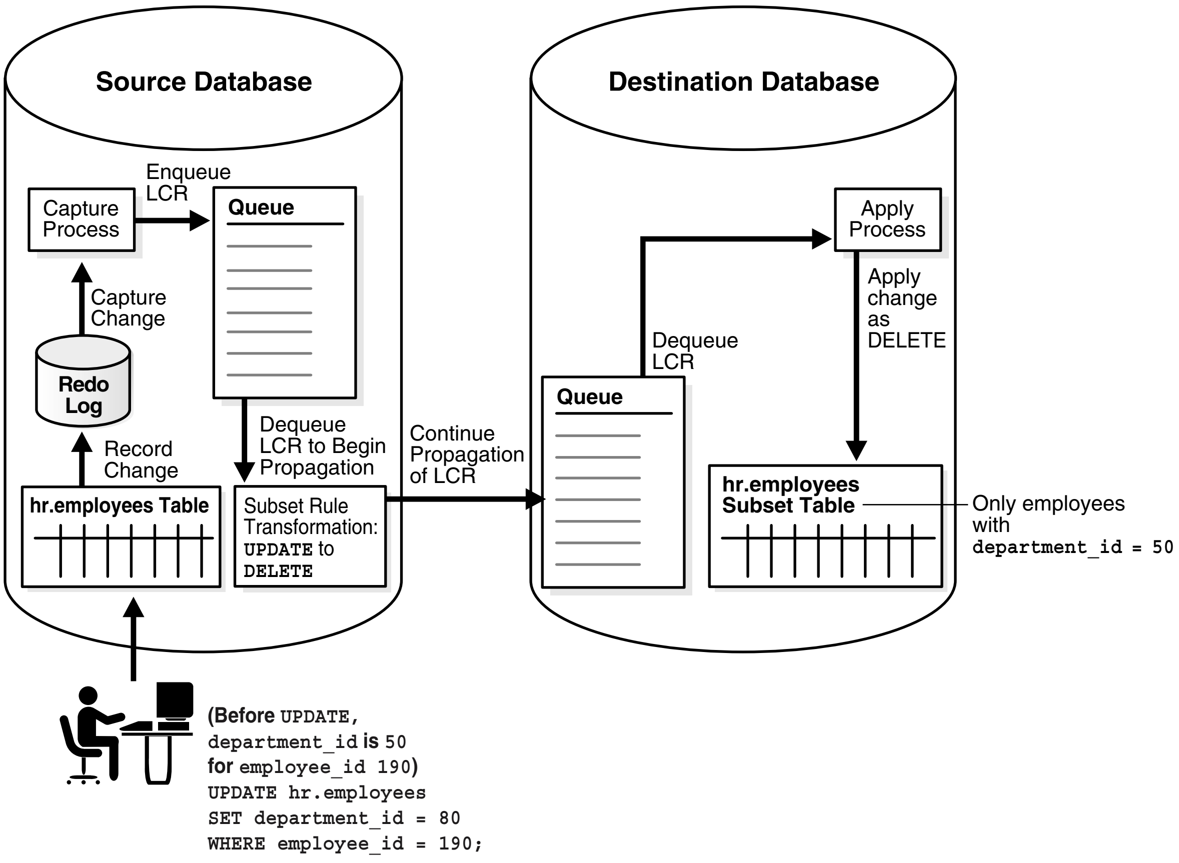


Similarly, if a captured update changes an employee's `department_id` from `50` to `20`, then a capture process with this subset rule converts the update operation into a `DELETE` operation.

Row Migration During Propagation When a subset rule is in the rule set for a propagation, an update operation may 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

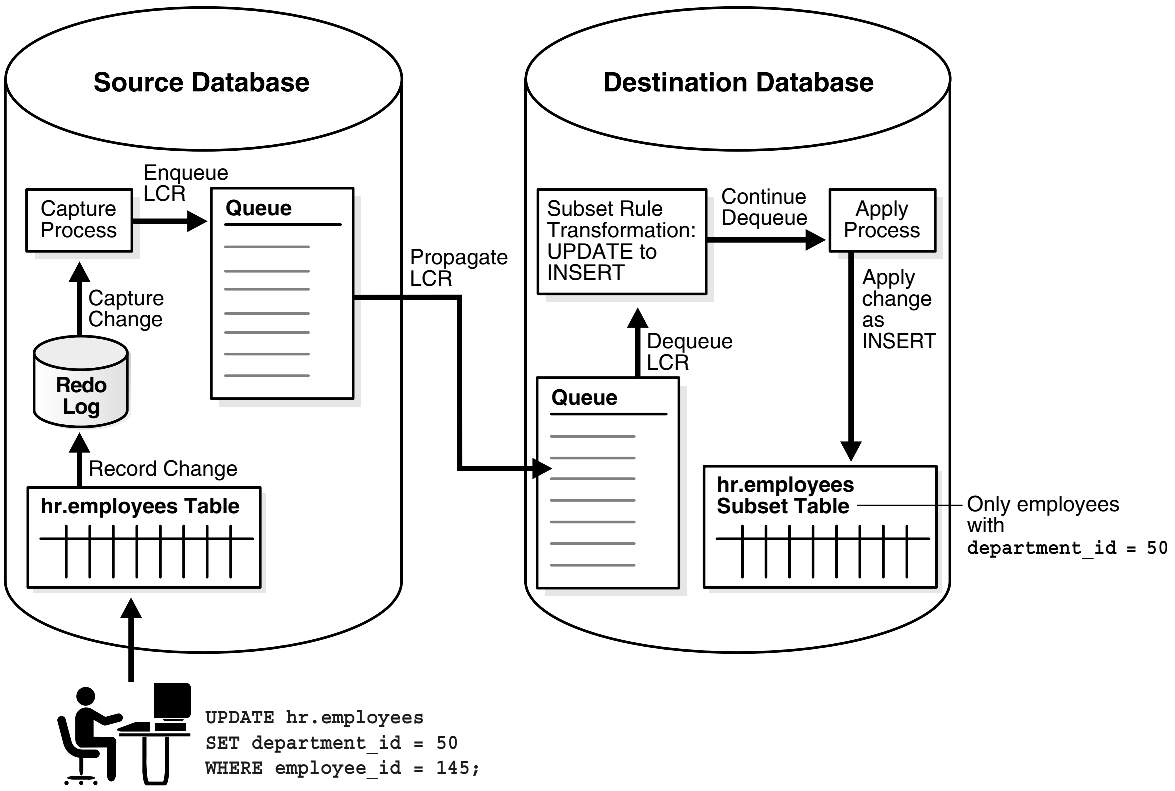


Similarly, if a captured update changes an employee's `department_id` from `80` to `50`, then a propagation with this subset rule converts the update operation into an `INSERT` operation.

Row Migration During Apply When a subset rule is in the rule set for an apply process, an update operation may 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

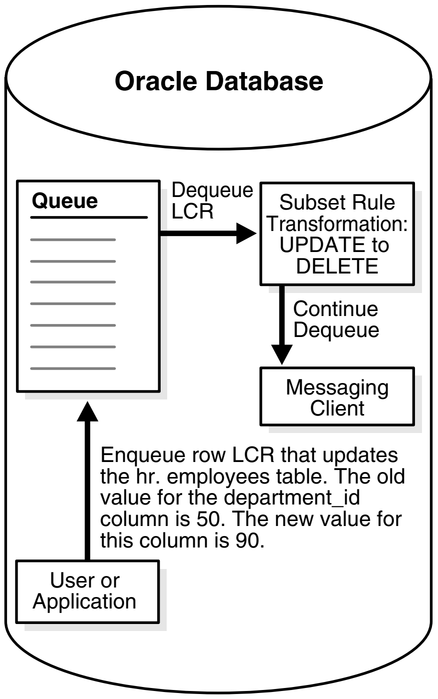


Similarly, if a captured update changes an employee's `department_id` from `50` to `20`, then an apply process with this subset rule converts the update operation into a `DELETE` operation.

Row Migration During Dequeue by a Messaging Client When a subset rule is in the rule set for a messaging client, an update operation may 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 may process this row LCR in any customized way. For example, it may 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


Oracle Database
Queue
Dequeue LCR
Subset Rule Transformation: UPDATE to DELETE
Continue Dequeue
Messaging Client
Enqueue row LCR that updates the hr. employees table. The old value for the department_id column is 50. The new value for this column is 90.
User or Application

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 certain cases, when a subset rule is specified, an update may be converted to an insert, and, in these cases, supplemental information may 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 Only Need 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 Only Need a Subset of Changes Subset rules should be used with a capture process when all destination databases of the capture process only need 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 only need 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 only need 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 networks environment, propagations may 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 may apply row LCRs that have been transformed by a row migration, then Oracle Corporation 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 may result.

For example, consider a scenario where 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 may 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 may 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 Corporation 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 in the positive rule set for a capture process, propagation, apply process, or messaging client:

- 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 rule's 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 rule's 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.

Message Rules

When you use a rule to specify a Streams task that is relevant only for a user-enqueued event of a specific message type, you are specifying a message rule. You can specify message rules for propagations, apply processes, and messaging clients.

A single message rule in the positive rule set for a propagation means that the propagation propagates the user-enqueued events 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 events 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 events of the message type that satisfy the rule condition. The apply process then sends these user-enqueued events 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 events 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 events 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 events of the message type in its queue that satisfy the rule condition. Unlike propagations and apply processes, which propagate or apply events automatically when they are running, a messaging client does not automatically dequeue or discard events. Instead, a messaging client must be used by a user or application to dequeue or discard events.

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 Events If region Is EUROPE and priority Is 1
- Send User-Enqueued Events 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 events of the following type:

```
CREATE TYPE strmadmin.region_pri_msg AS OBJECT(
  region         VARCHAR2(100),
  priority       NUMBER,
  message        VARCHAR2(3000))
/
```

Dequeue User-Enqueued Events 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 6-45

Send User-Enqueued Events 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 6-45

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's 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's condition and sends these messages to its message handler or re-enqueues the messages into a specified queue.

See Also:

- "Non-LCR User Message Processing" on page 4-7
- "Enqueue Destinations for Events During Apply" on page 6-52

System-Created Rules and Negative Rule Sets

You add system-created rules 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 events in its source queue that satisfy the rule.
- An apply process discards events in its queue that satisfy the rule.
- A messaging client discards events 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 also can 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 may 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 if you avoid using complex rules.

See Also:

- "Complex Rule Conditions" on page 6-58
- "System-Created Rules with Added User-Defined Conditions" on page 6-44 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 Events" on page 6-4

System-Created Rules with Added User-Defined Conditions

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

- *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

This section describes the system-created evaluation contexts used in Streams.

Evaluation Context for Global, Schema, Table, and Subset Rules

When you create global, schema, table, and subset rules, the system-created rule sets 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 may be used by capture processes, propagations, apply processes, and messaging clients.

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'));
  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

The ROW_VARIABLE_VALUE_FUNCTION converts a SYS.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 a SYS.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 events. A capture process enqueues row LCRs and DDL LCRs into its queue, and this function enables it to enqueue other internal events 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`

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.

Attention: Information about these internal functions is provided for reference purposes only. You should never run any of these functions directly.

See Also: *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 may 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 may 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 a `SYS.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 a `SYS.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. Also, 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 6-36
- "Evaluation Context for Global, Schema, Table, and Subset Rules" on page 6-45

Streams and Event Contexts

In Streams, capture processes and messaging clients do not use event contexts, but propagations and apply processes do. Both captured events and user-enqueued events can be staged in a queue. When an event is staged in a queue, a propagation or apply process can send the event, 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 Streams event itself as the value.

If you create a custom evaluation context, then you can create propagation and apply process rules 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 the event itself. You also can pass the event context to an evaluation function and a variable method function.

See Also:

- "Rule Set Evaluation" on page 5-13 for more information about event contexts
- "Explicit and Implicit Variables" on page 5-7 for more information about variable value functions
- "Evaluation Function" on page 5-9

Streams and Action Contexts

The following sections describe the purposes of action contexts 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
- User-Defined Rule-Based Transformations
- Enqueue Destinations for Events During Apply
- Execution Directives for Events During Apply

A different name-value pair may exist in a rule's action context 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 by the name-value pairs are performed in the following order:

1. Perform subset transformation
2. Perform user-defined rule-based transformation
3. Follow execution directive and perform execution if directed to do so (apply only)
4. 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 clients ignore the rule's action context.

Internal LCR Transformations in Subset Rules

When you use subset rules, an update operation may 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 a subset rule's 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 6-23
- "Managing Rule-Based Transformations" on page 12-18 for information about using rule-based transformation with subset rules

User-Defined Rule-Based Transformations

A rule-based transformation is any user-defined modification to an event that results when a rule evaluates to `TRUE`. The name-value pair for a user-defined rule-based transformation has `STREAMS$_TRANSFORM_FUNCTION` for the name and the name of the transformation function for the value.

See Also:

- "Rule-Based Transformations" on page 6-63
- "Managing Rule-Based Transformations" on page 12-18

Execution Directives for Events During Apply

The SET_EXECUTE procedure in the DBMS_APPLY_ADM package specifies whether an event 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 event. If an event that satisfies a rule should be executed by an apply process, then this name-value pair is not present in the rule's action context.

See Also: "Specifying Execute Directives for Apply Processes" on page 11-23

Enqueue Destinations for Events During Apply

The SET_ENQUEUE_DESTINATION procedure in the DBMS_APPLY_ADM package sets the queue where an event 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 Event Enqueues by Apply Processes" on page 11-21

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 rules 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 there are two rules that 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 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 may or may not occur, depending on which rule is returned.

You may 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: "Rule-Based Transformations" on page 6-63

Action Context Considerations for Schema and Global Rules

If you use an action context for a rule-based transformation, enqueue destination, or execute directive with a schema or global rule, then the action specified by the action context is carried out on an event if the event causes the schema or global rule to evaluate to `true`. For example, if a schema rule has an action context that specifies a rule-based transformation, then the transformation is performed on LCRs for the tables in the schema.

You may 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 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 may 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 6-7 for more information about schema and global rules

User-Created Rules, Rule Sets, and Evaluation Contexts

The DBMS_STREAMS_ADM package generates system-created rules and rule sets, and it may 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 he DBMS_RULE_ADM package to create them.

Some of the reasons you may need to use the DBMS_RULE_ADM package are the following:

- You need to create rules with rule conditions 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 may be a positive or negative rule set for a Streams client, and a rule set may be a positive rule set for one Streams client and a negative rule set for another.

This section contains the following topics:

- User-Created Rules and Rule Sets
- User-Created Evaluation Contexts

See Also:

- "Specifying a Rule Set for a Capture Process" on page 9-27
- "Specifying the Rule Set for a Propagation" on page 10-14
- "Specifying the Rule Set for an Apply Process" on page 11-11

User-Created Rules and Rule Sets

The following sections describe some of the types of rules and rule sets 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
- Avoid Using :dml and :ddl 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 might be easier than creating rules with the `DBMS_RULE_ADM` package.

See Also: "System-Created Rules with Added User-Defined Conditions" on page 6-44 for more information about the `and_condition` parameter

Rule Conditions for Specific Types of Operations

In some cases, you may want to capture, propagate, apply, or dequeue changes that contain only certain types of operations. For example, you may 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 may not be a simple rule condition, and non-simple rules may 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 5-4
- "Complex Rule Conditions" on page 6-58

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 and the COMPATIBLE_10_1 function in the DBMS_STREAMS package. These functions return constant values that correspond to 9.2.0 and 10.1.0 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 release 9.2 of Oracle.

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 release 10.1 or lower of Oracle. That is, the Streams client processes any row LCR that is compatible with release 9.2 or release 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 or global rules 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 14-74
- "Global Rules Example" on page 6-14, "Schema Rule Example" on page 6-18, and "System-Created Rules with Added User-Defined Conditions" on page 6-44
- *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 5-4. 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 6-9 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 are a wide range of complex rule conditions. The following sections contain some examples of complex rule conditions.

Note:

- Complex rule conditions may degrade rule evaluation performance.
- In rule conditions, names of database objects, such as tables and users, must exactly match the names in the database, including the case of each character. Also, the name cannot be enclosed in double quotes.
- 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 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.

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 6-39

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 certain 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 clients. In contrast, for non-Streams clients of the rules engine, an undefined variable in a rule condition may cause the rules engine to return `maybe_rules` to the client. When a rule set is evaluated, `maybe_rules` are rules that may evaluate to `TRUE` given more information.

The number of `maybe_rules` returned to Streams clients is reduced by treating each undefined variable as `NULL`, and reducing the number of `maybe_rules` can improve performance if it results in more efficient evaluation of a rule set when an event 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 event. So, for example, if the positive rule set for a capture process has such a rule, then the capture process may capture events 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 an event 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 an event 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 5-13

Avoid Using :dml and :ddl Variables as Function Parameters in Rule Conditions

Oracle Corporation 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 2-45

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 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

Rule-Based Transformations

In Streams, a **rule-based transformation** is any user-defined modification to an event that results when a rule in a positive rule set evaluates to `TRUE`. You may use a rule-based transformation to modify both captured and user-enqueued events, and these events may be LCRs or user messages. A transformation must be defined as a PL/SQL function that takes a `SYS.AnyData` object as input and returns a `SYS.AnyData` object. Rule-based transformations support only one to one transformations.

For example, a rule-based transformation may be used when the datatype of a particular column in a table is different at two different databases. Such a column could be a `NUMBER` column in the source database and a `VARCHAR2` column in the destination database. In this case, the transformation takes as input a SYS.AnyData object containing a row LCR with a NUMBER datatype for a column and returns a SYS.AnyData object containing a row LCR with a VARCHAR2 datatype for the same column.

Other examples of transformations on events include:

- Renaming the owner of a database object
- Renaming a database object
- Renaming or removing a column
- 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

Although you can modify a captured LCR with a rule-based transformation, a rule-based transformation that is executed on a captured LCR must not construct a new LCR and return it. That is, a rule-based transformation must return the same captured LCR that it receives. However, a rule-based transformation that receives a user-enqueued event may construct a new event and return it. In this case, the returned event may be an LCR constructed by the rule-based transformation.

You use the SET_RULE_TRANSFORM_FUNCTION procedure in the DBMS_STREAMS_ADM package to specify a rule-based transformation for a rule. This procedure modifies the rule's action context 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 an event. 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 rule-based transformation in Streams always consists of the following name-value pair in an action context:

- The name is STREAMS$_TRANSFORM_FUNCTION.
- The value is a SYS.AnyData instance containing a PL/SQL function name specified as a VARCHAR2. This function performs the transformation.

You can view the existing rule-based transformations in a database by querying the DBA_STREAMS_TRANSFORM_FUNCTION data dictionary view.

The user that 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 user who calls the transformation function is the capture user for the capture process.
- If a transformation is specified for a rule used by a propagation, then the user who calls the transformation function is the owner of the source queue for the propagation.
- If a transformation is specified on a rule used by an apply process, then the user who calls the transformation function is the apply user for the apply process.
- If a transformation is specified on a rule used by a messaging client, then the user who calls the transformation function is the user who invokes the messaging client.

When a rule in a positive rule set evaluates to TRUE for an event in a Streams environment, and an action context that contains a name-value pair with the name STREAMS$_TRANSFORM_FUNCTION is returned, the PL/SQL function is run, taking the event as an input parameter. Other names in an action context beginning with STREAMS$_ 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 an event 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 may be associated with different operation types, tables, or schemas for which events are being captured, propagated, applied, or dequeued.

The following are considerations for rule-based transformations:

- 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 section 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 transformations on row LCRs that are expensive, then consider making these modifications within a DML handler instead, because DML handlers can execute in parallel when apply parallelism is greater than 1.
- When you perform 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.
- You cannot use a rule-based transformation to convert an LCR event into a non-LCR event. This restriction applies to captured LCRs and user-enqueued LCRs.

The following sections provide more information about rule-based transformations for each type of Streams client:

- Rule-Based Transformations and a Capture Process
- Rule-Based Transformations and a Propagation
- Rule-Based Transformations and an Apply Process
- Rule-Based Transformations and a Messaging Client
- Multiple Rule-Based Transformations

See Also:

- "Managing Rule-Based Transformations" on page 12-18
- "Rule Action Context" on page 5-10
- *Oracle Streams Advanced Queuing User's Guide and Reference* and *PL/SQL Packages and Types Reference* for more information about the `DBMS_TRANSFORM` package
- "Event Processing with an Apply Process" on page 4-3 for more information about DML handlers

Rule-Based Transformations and a Capture Process

If a capture process uses a positive rule set, then both of the following conditions must be met, in order, for a transformation to be performed during capture:

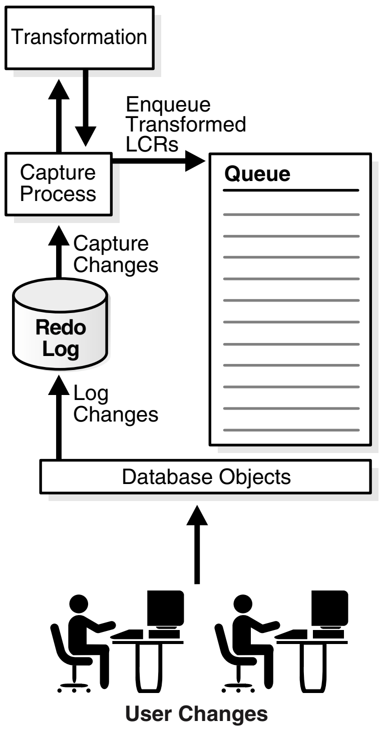
- A rule in the positive rule set evaluates to `TRUE` for a particular change found in the redo log.
- An action context containing a name-value pair with the name `STREAMS$_TRANSFORM_FUNCTION` is returned to the capture process when the rule is evaluated.

Given these conditions, the capture process completes the following steps:

1. Formats the change in the redo log into an LCR
2. Converts the LCR into a `SYS.AnyData` object
3. Runs the PL/SQL function in the name-value pair to transform the `SYS.AnyData` object
4. Enqueues the transformed `SYS.AnyData` object into the queue associated with the capture process

All actions are performed as the capture user. Figure 6–6 shows a transformation during capture.

Figure 6–6 Transformation During Capture



For example, if an LCR event is transformed during capture, then the transformed LCR event is enqueued into the queue used by the capture process. Therefore, if such a captured LCR event 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 event 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 may 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 event, because the transformation is performed only once at the source, not at multiple destinations.

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 event.

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 "Datatypes Captured" on page 2-8.

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 a Propagation

If a propagation uses a positive rule set, then both of the following conditions must be met, in order, for a transformation to be performed during propagation:

- A rule in the positive rule set evaluates to `TRUE` for an event in the source queue for the propagation. This event can be a captured or a user-enqueued event.
- An action context containing a name-value pair with the name `STREAMS$_TRANSFORM_FUNCTION` is returned to the propagation when the rule is evaluated.

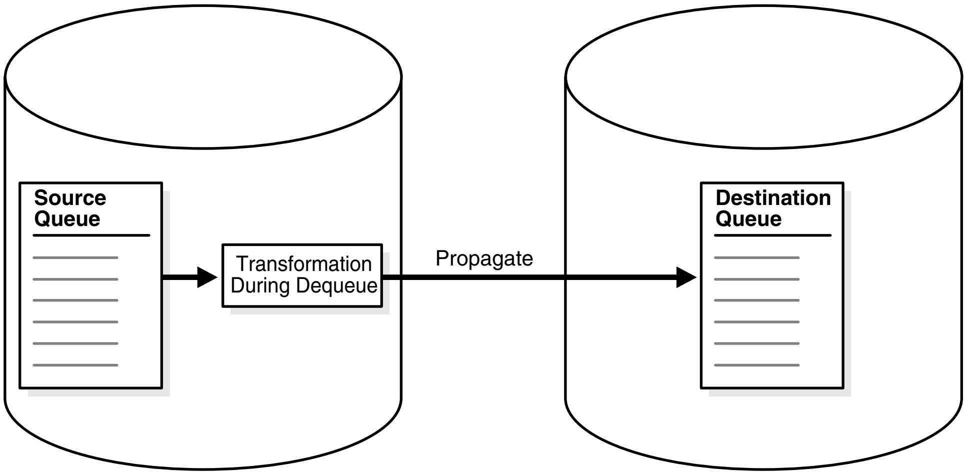
See Also: "Captured and User-Enqueued Events" on page 3-3

Given these conditions, the propagation process includes the following steps:

1. The propagation starts dequeuing the event from the source queue.
2. The source queue owner runs the PL/SQL function in the name-value pair to transform the event.
3. The propagation completes dequeuing the transformed event.
4. The propagation propagates the transformed event to the destination queue.

Figure 6–7 shows a transformation during propagation.

Figure 6–7 Transformation During Propagation



For example, suppose you use a rule-based transformation for a propagation that propagates events from the `dbs1.net` database to the `dbs2.net` database, but you do not use a rule-based transformation for a propagation that propagates events from the `dbs1.net` database to the `dbs3.net` database.

In this case, an event in the queue at `dbs1.net` can be transformed before it is propagated to `dbs2.net`, but the same event 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 event, and the queue at `dbs3.net` contains the original event.

The advantages of performing transformations during propagation are the following:

- Security can be improved if the transformation removes or changes private information before events are propagated.
- Some destination queues can receive a transformed event, while other destination queues can receive the original event.
- Different destinations can receive different variations of the same event.

The possible disadvantages of performing transformations during propagation are the following:

- Once an event is transformed, any database to which it is propagated after the first propagation receives the transformed event. For example, if `dbs2.net` propagates the event to `dbs4.net`, then `dbs4.net` receives the transformed event.
- When the first propagation in a directed network performs the transformation, and the capture process that captured the event 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 may be done multiple times when multiple destination databases need the same transformation.

Rule-Based Transformation Errors During Propagation

If an error occurs when the transformation function is run during propagation, then the event that caused the error is not dequeued or propagated, and the error is returned to the propagation. Before the event can be propagated, you must change or remove the rule-based transformation to avoid the error.

Rule-Based Transformations and an Apply Process

If an apply process uses a positive rule set, then both of the following conditions must be met, in order, for a transformation to be performed during apply:

- A rule in the positive rule set evaluates to `TRUE` for an event in the queue associated with the apply process. This event can be a captured or a user-enqueued event.

- An action context containing a name-value pair with the name `STREAMS$_TRANSFORM_FUNCTION` is returned to the apply process when the rule is evaluated.

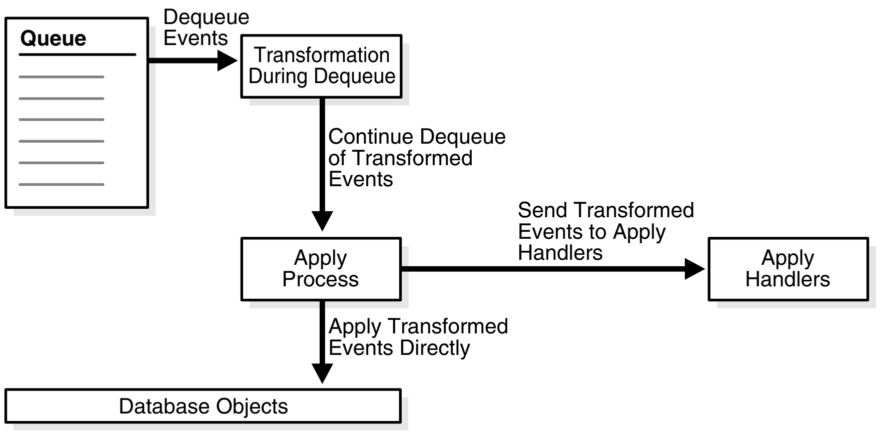
See Also: "Captured and User-Enqueued Events" on page 3-3

Given these conditions, the apply process completes the following steps:

1. Starts to dequeue the event from the queue
2. Runs the PL/SQL function in the name-value pair to transform the event during dequeue
3. Completes dequeuing the transformed event
4. Applies the transformed event, which may involve changing database objects at the destination database or sending the transformed event to an apply handler

All actions are performed as the apply user. Figure 6–8 shows a transformation during apply.

Figure 6–8 Transformation During Apply


Queue
Dequeue Events
Transformation During Dequeue
Continue Dequeue of Transformed Events
Send Transformed Events to Apply Handlers
Apply Process
Apply Handlers
Apply Transformed Events Directly
Database Objects

For example, suppose an event is propagated from the `dbs1.net` database to the `dbs2.net` database in its original form. When the apply process dequeues the event from a queue at `dbs2.net`, the event is transformed.

The possible advantages of performing transformations during apply are the following:

- Any database to which the event is propagated after the first propagation can receive the event in its original form. For example, if `dbs2.net` propagates the event to `dbs4.net`, then `dbs4.net` can receive the original event.
- 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 may be a concern if the events contain private information, because all databases to which the events are propagated receive the original events.
- The same transformation may 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 event that caused the error is not dequeued, the transaction containing the event 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 Events

If an apply error occurs for a transaction in which some of the events have been transformed by a rule-based transformation, then the transformed events are moved to the error queue with all of the other events 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 events, then the transformation is not performed on the events again because the apply process rule set containing the rule is not evaluated again.

Rule-Based Transformations and a Messaging Client

If a messaging client uses a positive rule set, then both of the following conditions must be met, in order, for a transformation to be performed when a messaging client dequeues an event from its queue:

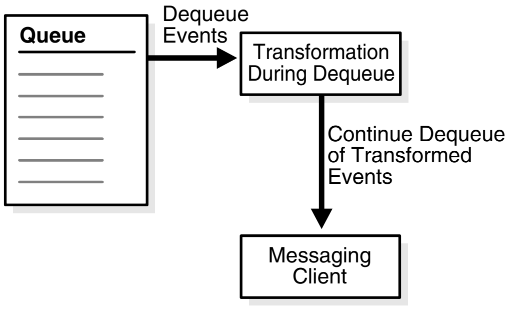
- A rule in the positive rule set of the messaging client evaluates to `TRUE` for the event. This event can be a user-enqueued LCR or a user-enqueued message.
- An action context containing a name-value pair with the name `STREAMS$_TRANSFORM_FUNCTION` is returned to the messaging client when the rule is evaluated.

Given these conditions, the messaging client completes the following steps:

1. Starts to dequeue the event from the queue
2. Runs the PL/SQL function in the name-value pair to transform the event during dequeue
3. Completes dequeuing the transformed event

All actions are performed as the user who invokes the messaging client. Figure 6–9 shows a transformation during messaging client dequeue.

Figure 6–9 Transformation During Messaging Client Dequeue


Queue
Dequeue Events
Transformation During Dequeue
Continue Dequeue of Transformed Events
Messaging Client

For example, suppose an event is propagated from the `dbs1.net` database to the `dbs2.net` database in its original form. When the messaging client dequeues the event from a queue at `dbs2.net`, the event is transformed.

One possible advantage of performing transformations during dequeue in a messaging environment is that any database to which the event is propagated after the first propagation can receive the event in its original form. For example, if `dbs2.net` propagates the event to `dbs4.net`, then `dbs4.net` can receive the original event.

The possible disadvantages of performing transformations during dequeue in a messaging environment are the following:

- Security may be a concern if the events contain private information, because all databases to which the events are propagated receive the original events.
- The same transformation may be done multiple times when multiple destination databases 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 event that caused the error is not dequeued, and the error is returned to the messaging client. Before the event 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 an event 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 an event during capture. If some recipients require additional custom transformations, then you can transform the previously transformed event during propagation, apply, or dequeue.

7 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 may 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 may 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 Real Application Clusters Administrator's Guide*

Protection from Failures

Oracle 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 may 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.

Some of the benefits of a logical copy include the following:

- 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 may choose to use Streams to configure and maintain a logical copy of your production database. Although using Streams may require additional work, it offers increased flexibility that may 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 may 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 may be left at the primary database, but this data will be automatically applied when the primary database recovers. This ability may 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 where you may 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 also can 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 re-analyzed 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 may 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 3-8

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 events 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 Corporation recommends Oracle 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 Corporation 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 Oracle RAC with Streams introduces some important considerations. When running in an Oracle 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 queues. An apply process runs on the instance that owns the queue from which the apply process dequeues its events. That may or may not be the same queue on which capture runs.

Any propagation to the database running Oracle 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 events.

You may choose to use a cold failover cluster to protect from system failure rather than Oracle RAC. A cold failover cluster is not an Oracle Real Application Cluster. 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 2-27
- "SYS.AnyData Queues and Oracle Real Application Clusters" on page 3-14
- "Streams Apply Processes and Oracle Real Application Clusters" on page 4-13

Local or Downstream Capture with Streams

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 may not have been captured. With local capture, those changes may not be available until the source database is recovered. In the event of a catastrophic failure, those changes may 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 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 2-17

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 will automatically return 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 will restart automatically.

Similarly, for a capture process running in an Oracle RAC environment, if an instance running the capture process fails, then the queue that receives the captured LCRs will be assigned to another node in the cluster, and the capture process will be restarted automatically. A capture process will follow 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 2-27
- "Starting a Capture Process" on page 9-26
- "SYS.AnyData Queues and Oracle Real Application Clusters" on page 3-14 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 may 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 an Oracle 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 queues for propagations, then 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 Network Connectivity and Database Links" on page 8-11 for information about creating database links in a Streams environment
- "SYS.AnyData Queues and Oracle Real Application Clusters" on page 3-14 for more information about database links in a RAC environment

Propagation Job Restart After a Failover

For events 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 will resume when the single database instance is restarted.

When running in an Oracle RAC environment, a propagation job runs on the instance that owns the source queue from which the propagation job sends events 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: "SYS.AnyData Queues and Oracle Real Application Clusters" on page 3-14

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 will automatically return 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 will restart automatically.

Similarly, in an Oracle RAC cluster, if an instance hosting the apply process fails, then the queue from which the apply process dequeues events will be assigned to another node in the cluster, and the apply process will be restarted automatically. An apply process will follow 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 4-13
- "Starting an Apply Process" on page 11-10
- "SYS.AnyData Queues and Oracle Real Application Clusters" on page 3-14 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:

8 Preparing a Streams Environment

This chapter provides instructions for preparing a database or a distributed database environment to use Streams and for configuring a Streams environment.

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: To ensure security, use a password other than strmadminpw for the Streams administrator.

4. Grant the Streams administrator CONNECT and RESOURCE role so that this administrator can connect to the database and manage different types of database objects in the administrator's own schema. Also, grant the Streams administrator DBA role.

```
GRANT CONNECT, RESOURCE, DBA TO strmadmin;
```

5. Optionally, run the GRANT_ADMIN_PRIVILEGE procedure in the DBMS_STREAMS_AUTH package. You may 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 may run the GRANT_ADMIN_PRIVILEGE procedure to grant such privileges to the Streams administrator, or you may 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: *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:

   - `SELECT_CATALOG_ROLE` if you want to grant the user privileges to query non-Streams data dictionary views
   - `SELECT ANY DICTIONARY` privilege if you plan to use the Streams tool in the Oracle Enterprise Manager Console

- 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 may be used in apply handlers or error handlers. 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 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 events into or dequeue events from the queue. The Streams administrator may also need `ENQUEUE` or `DEQUEUE` privileges on the queue, or both. See "Enabling a User to Perform Operations on a Secure Queue" on page 10-3 for instructions.
- `EXECUTE` privilege on any object types that the Streams administrator may 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 14-4

Setting Initialization Parameters Relevant to Streams

Table 8–1 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 SESSION` or `ALTER SYSTEM` statement while an instance is running.

> **See Also:** *Oracle Database Reference* for more information about these initialization parameters

Table 8–1 Initialization Parameters Relevant to Streams

Parameter	Values	Description
`COMPATIBLE`	**Default:** `9.2.0` **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*, 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.
`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 `J`*`n`* 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 events 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 8–1 Initialization Parameters Relevant to Streams (Cont.)

Parameter	Values	Description
`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 log files to the downstream database using log transport services, at least one log archive destination must be at the site running the downstream capture process. **See Also:** *Oracle Data Guard Concepts and Administration*
`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 log files to the downstream database using log 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 from the values of the following parameters:
	`CPU_COUNT`
	`PARALLEL_ADAPTIVE_MULTI_USER`
	`PARALLEL_AUTOMATIC_TUNING`
	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 may use multiple parallel execution servers. Set this initialization parameter to an appropriate value to ensure that there are enough parallel execution servers.

Table 8–1 Initialization Parameters Relevant to Streams (Cont.)

Parameter	Values	Description
`PROCESSES`	**Default:** Derived from `PARALLEL_MAX_SERVERS` **Range:** `6` to operating system dependent limit **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.
`REMOTE_ARCHIVE_ENABLE`	**Default:** `true` **Range**: `true` or `false` **Modifiable?:** No	Enables or disables the sending of redo archival to remote destinations and the receipt of remotely archived redo. To use downstream capture and copy the redo log files to the downstream database using log transport services, this parameter must be set to `true` at both the source database and the downstream database.
`SESSIONS`	**Default:** Derived from: `(1.1 * PROCESSES) + 5` **Range:** `1` to 2^{31} **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 may 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 limit **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 may need to increase the size of this parameter.
`SHARED_POOL_SIZE`	**Default:** 32-bit platforms: `32 MB`, rounded up to the nearest granule size 64-bit platforms: `84 MB`, rounded up to the nearest granule size **Range:** Minimum: the granule size Maximum: 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 `STREAMS_POOL_SIZE` initialization parameter is set to zero, then Streams may use up to 10% of the shared pool.

Table 8–1 Initialization Parameters Relevant to Streams (Cont.)

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 captured events. In addition, the Streams pool is used for internal communications during parallel capture and apply. If the size of the Streams pool is greater than zero, then any SGA memory used by Streams is allocated from the Streams pool. If the Streams pool size is set to zero, then SGA memory used by Streams is allocated from the shared pool and may use up to 10% of the shared pool. This parameter is modifiable. However, if this parameter is set to zero when an instance starts, then increasing it beyond zero has no effect on the current instance because it is using the shared pool for Streams allocations. Also, if this parameter is set to a value greater than zero when an instance starts and is then reduced to zero when the 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 ■ 1 MB for each apply process parallelism ■ 10 MB or more for each queue staging captured events For example, if parallelism is set to `3` for a capture process, then increase the Streams pool by 30 MB. If parallelism is set to 5 for an apply process, then increase the Streams pool by `5` MB.

Table 8–1 Initialization Parameters Relevant to Streams (Cont.)

Parameter	Values	Description
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 Administrator's Guide* for more information about the retention period and the undo tablespace

Preparing a Database to Run a Streams Capture Process

Any source database that generates redo log information that will be captured by a capture process must be running in ARCHIVELOG mode. In addition, make sure the initialization parameters are set properly on any database that will run a capture process.

See Also:

- "ARCHIVELOG Mode and a Capture Process" on page 2-46
- *Oracle Database Administrator's Guide* for information about running a database in ARCHIVELOG mode
- "Setting Initialization Parameters Relevant to Streams" on page 8-6

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 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 events 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 events 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

9 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 a `SYS.AnyData` queue.

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 2, "Streams Capture Process"
- "Configuring a Streams Administrator" on page 8-2

Creating a Capture Process

You can create a capture process that captures changes to the local source database, or you can create a capture process that captures changes remotely at a downstream database. If a capture process runs on a downstream database, then redo log files from the source database are copied to the downstream database, and the capture process captures changes in these redo log files 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 may add table, schema, or global rules 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 may 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 "Capture Process Creation" on page 2-32 for more information about the first SCN and start SCN for a capture process.

The following tasks must be completed before you create a capture process:

- Complete the tasks described in "Preparing a Database to Run a Streams Capture Process" on page 8-11.
- Create a `SYS.AnyData` queue to associate with the capture process, if one does not exist. See "Creating a SYS.AnyData Queue" on page 10-2 for instructions.

The following sections describe:

- Creating a Local Capture Process
- Creating a Downstream Capture Process That Assigns Log Files Implicitly
- Creating a Downstream Capture Process That Assigns Log Files 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 2-32
- "First SCN and Start SCN" on page 2-24
- *Oracle Streams Replication Administrator's Guide* for information about changing the DBID or global name of a source database

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.

Example of Creating a Local Capture Process Using DBMS_STREAMS_ADM

The following is an example that 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         => 'strm01_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 strm01_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 specified by the system.
- Creates two rules. 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 specified by the system.
- 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 rules 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 also may 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 2-32
- "System-Created Rules" on page 6-7
- *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 is an example that 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         => 'strm02_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 strm02_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 also may 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 2-32
- "SCN Values Relating to a Capture Process" on page 2-23

Creating a Downstream Capture Process That Assigns Log Files Implicitly

To create a capture process that performs downstream capture, you must use the `CREATE_CAPTURE` procedure. The following sections describe creating a downstream capture process that uses a database link to the source database and one that does not. In both examples, 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 `strm03_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 log transport services or manually from the source database to the downstream database.

Preparing to Copy Redo Log Files for 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. Configure Oracle Net so that the source database can communicate with the downstream database.

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

2. Set the following initialization parameters to configure log transport services to copy 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.

     – `NOREGISTER` - Specify this attribute so that the downstream database location is not recorded in the downstream database control file.

     – `REOPEN` - Specify the minimum number of seconds the archiver process (`ARC`*n*) should wait before trying to access the downstream database location if a previous attempt to access this location failed.

     – `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 is an example of an `LOG_ARCHIVE_DEST_n` setting that specifies a downstream database:

```
LOG_ARCHIVE_DEST_2='SERVICE=DBS2.NET ARCH OPTIONAL NOREGISTER REOPEN=60
   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.

- 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 both the source database and the downstream database, set the `REMOTE_ARCHIVE_ENABLE` initialization parameter to `true`.

  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 may 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 downstream database because the global name of the source database is sent to the downstream database only if the source database is open.

4. Specify primary key supplemental logging for the `hr.departments` table at the source database `dbs1.net`:

   ```
   ALTER TABLE hr.departments ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY) COLUMNS;
   ```

Primary key supplemental logging is required for the hr.departments table because the examples in this section create capture processes that capture changes to this table.

5. Perform the steps in one of the following sections depending on whether or not you want to use a database link from the downstream database to the source database:

   - "Creating a Downstream Capture Process That Uses a Database Link" on page 9-9
   - "Creating a Downstream Capture Process That Does Not Use a Database Link" on page 9-13

Creating a Downstream Capture Process That Uses a Database Link

This section contains an example that runs the CREATE_CAPTURE procedure in the DBMS_CAPTURE_ADM package to create a 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.

Connect to the downstream database dbs2.net as the Streams administrator and complete the following steps:

1. 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:

```
CONNECT strmadmin/strmadminpw@dbs2.net

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.

2. Create the queue for the capture process if it does not exist:

```
BEGIN
  DBMS_STREAMS_ADM.SET_UP_QUEUE(
    queue_table  => 'strmadmin.strm03_queue_table',
    queue_name   => 'strmadmin.strm03_queue');
END;
/
```

3. Run the `CREATE_CAPTURE` procedure to create the capture process:

```
BEGIN
  DBMS_CAPTURE_ADM.CREATE_CAPTURE(
    queue_name         => 'strm03_queue',
    capture_name       => 'strm03_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 `strm03_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 `strm03_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 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 can be added to the capture process automatically using log 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 `strm03_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 `strm03_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 `strm03_capture` 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:

- "SCN Values Relating to a Capture Process" on page 2-23
- *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. 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        =>  'strm03_capture',
    queue_name          =>  'strm03_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 `strm03_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 1

Now you can configure propagation or apply, or both, of the LCRs captured by the `strm03_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` 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 `strm03_queue` queue. Setting the instantiation SCN for an object at a database is required only if an apply process applies LCRs for the object.

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 Downstream Capture Process That Does Not Use a Database Link

This section contains an example that runs the `CREATE_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package to create a capture process at the `dbs2.net` downstream database that captures changes made to the `dbs1.net` source database. The capture process in this example does not use a database link to `dbs1.net`.

Complete the following steps:

1. 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.

2. 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.

If you choose to build the data dictionary without displaying the valid first SCN value when the procedure completes, then you can query the `V$ARCHIVED_LOG` dynamic performance view to determine a valid first SCN value for the capture process that will be created at `dbs2.net`.

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

Your output looks similar to the following:

```
FIRST_CHANGE#
-------------
       409391
```

If more than one value is returned, then make a note of the highest value.

3. Prepare the `hr.departments` table for instantiation:

```
BEGIN
  DBMS_CAPTURE_ADM.PREPARE_TABLE_INSTANTIATION(
     table_name  =>  'hr.departments');
END;
/
```

4. 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
```

5. Create the queue for the capture process if it does not exist:

```
BEGIN
  DBMS_STREAMS_ADM.SET_UP_QUEUE(
    queue_table  => 'strmadmin.strm03_queue_table',
    queue_name   => 'strmadmin.strm03_queue');
END;
```

```
/
```

6. Run the `CREATE_CAPTURE` procedure to create the capture process and specify the value obtained in Step 2 for the `first_scn` parameter:

```
BEGIN
  DBMS_CAPTURE_ADM.CREATE_CAPTURE(
    queue_name         => 'strm03_queue',
    capture_name       => 'strm04_capture',
    rule_set_name      => NULL,
    start_scn          => NULL,
    source_database    => 'dbs1.net',
    use_database_link  => false,
    first_scn          => 409391, -- Use value from Step 2
    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 `strm03_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 2. 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 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 log 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.

See Also:

- "Capture Process Creation" on page 2-32
- "SCN Values Relating to a Capture Process" on page 2-23
- *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

7. 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          =>  'strm03_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 set the instantiation SCN for the `hr.departments` table at `dbs2.net` by running the following procedure at the source database `dbs1.net`:

```
CONNECT strmadmin/strmadminpw@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 `strm03_queue` queue. Setting the instantiation SCN for an object at a database is required only if an apply process applies LCRs for the object.

Note:

- To set the instantiation SCN using the previous example requires a database link from the source database to the destination database.
- 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 Downstream Capture Process That Assigns Log Files Explicitly

To create a capture process that performs downstream capture, you must use the `CREATE_CAPTURE` procedure. This section describes creating a 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 `strm03_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. Connect to the source database `dbs1.net` as the `hr` user:

   ```
   CONNECT hr/hr@dbs1.net
   ```

2. Specify primary key supplemental logging for the `hr.departments` table:

   ```
   ALTER TABLE hr.departments ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY) COLUMNS;
   ```

   Primary key supplemental logging is required for the `hr.departments` table because this example creates a capture processes that captures changes to this table.

3. 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.

4. 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.

5. Prepare the hr.departments table for instantiation:

   ```
   BEGIN
     DBMS_CAPTURE_ADM.PREPARE_TABLE_INSTANTIATION(
        table_name  =>  'hr.departments');
   END;
   /
   ```

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 2 for the `first_scn` parameter:

```
BEGIN
  DBMS_CAPTURE_ADM.CREATE_CAPTURE(
    queue_name         => 'strm03_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 2
    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 `strm03_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 2. 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 2-32
- "SCN Values Relating to a Capture Process" on page 2-23
- *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          =>  'strm03_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`.

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 4 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 4 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 by running the following procedure at the source database dbs1.net:

```
CONNECT strmadmin/strmadminpw@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 `strm03_queue` queue. Setting the instantiation SCN for an object at a database is required only if an apply process applies LCRs for the object.

Note:

- To set the instantiation SCN using the previous example requires a database link from the source database to the destination database.
- 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 processes 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.

Run the following procedure to create the capture process:

```
BEGIN
  DBMS_CAPTURE_ADM.CREATE_CAPTURE(
    queue_name         => 'strm01_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 `strm01_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 also may 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 log information 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 2-32
- "First SCN and Start SCN Specifications During Capture Process Creation" on page 2-40

Starting, Stopping, and Dropping a Capture Process

This section contains instructions for starting, stopping and dropping a capture process. It contains the following topics:

- Starting a Capture Process
- Stopping a Capture Process
- Dropping a Capture Process

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 may 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;
/
```

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 sets 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 and negative rule set for the capture process may be dropped. If this procedure drops a rule set, then it also drops any rules in the rule set that are not in another rule set.

Note: A capture process must be stopped before it can be dropped.

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
- Specifying 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 rules to a rule set for an existing capture process, you can run one of the following procedures and specify the existing 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`

Excluding the `ADD_SUBSET_RULES` procedure, these procedures can add rules to the positive 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 6-7

Adding Rules to the Positive Rule Set for a Capture Process

The following is an example that 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      =>  'strm01_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 specified by the system.
- 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. If the capture process is performing downstream capture, then the table is prepared for instantiation 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.

Adding Rules to the Negative Rule Set for a Capture Process

The following is an example that 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      =>  'strm01_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 specified by the system.
- 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.

In addition, 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 6-6

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, even if the parameter value is a number.

See Also:

- "Capture Process Parameters" on page 2-47
- The DBMS_CAPTURE_ADM.SET_PARAMETER procedure in the *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 sets. 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 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.

Specifying Supplemental Logging at a Source Database

Supplemental logging must be specified for certain 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 may be required in any environment that processes captured events 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 SCN Values for an Existing Capture Process

This section contains the following topics:

- Setting the First SCN for an Existing Capture Process
- Setting the Start SCN for an Existing Capture Process

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 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 prior 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 2-23
- "The LogMiner Data Dictionary for a Capture Process" on page 2-34
- "First SCN and Start SCN Specifications During Capture Process Creation" on page 2-40
- "Displaying the Redo Log Files That Will Never Be Needed by Any Capture Process" on page 14-13

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 databases 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. 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;
```

Also, when you reset a start SCN for a capture process, make sure the required redo log files are available to the capture process.

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 2-23
- *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 rules 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 a Downstream Capture Process That Does Not Use a Database Link" on page 9-13 created the capture process `strm04_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       => 'strm04_capture',
    use_database_link  => true);
END;
/
```

See Also: "Local Capture and Downstream Capture" on page 2-17

Managing Extra Attributes in Captured LCRs

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 also can 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 LCRs
- Excluding Extra Attributes from Captured LCRs

See Also:

- "Extra Information in LCRs" on page 2-6
- "Viewing the Extra Attributes Captured by Each Capture Process" on page 14-17
- *PL/SQL Packages and Types Reference* for more information about the INCLUDE_EXTRA_ATTRIBUTE procedure

Including Extra Attributes in Captured LCRs

To instruct a capture process named `strm01_capture` to include the transaction name in each captured LCR, 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 LCRs

To instruct a capture process named `strm01_capture` to exclude the transaction name from each captured LCR, run the following procedure:

```
BEGIN
  DBMS_CAPTURE_ADM.INCLUDE_EXTRA_ATTRIBUTE(
    capture_name   => 'strm01_capture',
    attribute_name => 'tx_name',
    include        => false);
END;
/
```

10

Managing Staging and Propagation

This chapter provides instructions for managing `SYS.AnyData` queues, propagations, and messaging environments.

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 3, "Streams Staging and Propagation"
- "Configuring a Streams Administrator" on page 8-2

Managing SYS.AnyData Queues

A `SYS.AnyData` queue stages events whose payloads are of `SYS.AnyData` type. Therefore, a `SYS.AnyData` queue can stage an event with payload of nearly any type, if the payload is wrapped in a `SYS.AnyData` wrapper. Each Streams capture process and apply process is associated with one `SYS.AnyData` queue, and each Streams propagation is associated with one Streams source queue and one `SYS.AnyData` destination queue.

This section provides instructions for completing the following tasks related to `SYS.AnyData` queues:

- Creating a SYS.AnyData Queue
- Enabling a User to Perform Operations on a Secure Queue
- Disabling a User from Performing Operations on a Secure Queue
- Removing a SYS.AnyData Queue

Creating a SYS.AnyData Queue

You use the `SET_UP_QUEUE` procedure in the `DBMS_STREAMS_ADM` package to create a `SYS.AnyData` queue. This procedure enables you to specify the following for the `SYS.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

This procedure creates a queue that is both a secure queue and a transactional queue and starts the newly created queue.

For example, to create a `SYS.AnyData` queue named `strm01_queue` in the `strmadmin` schema with a queue table named `strm01_queue_table` and grant the `hr` user the privileges necessary to enqueue events into and dequeue events from the queue, run the following procedure:

```
BEGIN
  DBMS_STREAMS_ADM.SET_UP_QUEUE(
    queue_table  => 'strmadmin.strm01_queue_table',
    queue_name   => 'strmadmin.strm01_queue',
    queue_user   => 'hr');
END;
/
```

You also can use procedures in the DBMS_AQADM package to create a SYS.AnyData queue.

Note:

- Queue names and queue table names can be a maximum of 24 bytes.
- An event cannot be enqueued into a queue unless a subscriber who can dequeue the event is configured.

See Also:

- "Wrapping User Message Payloads in a SYS.AnyData Wrapper and Enqueuing Them" on page 10-20 for an example that creates a SYS.AnyData queue using procedures in the DBMS_AQADM package
- "Secure Queues" on page 3-19
- "Transactional and Nontransactional Queues" on page 3-22

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 a SYS.AnyData Queue" on page 10-2. 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 events 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 3-19
- *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 may 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 10-3, 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 a SYS.AnyData Queue" on page 10-2.

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 3-19
- *PL/SQL Packages and Types Reference* for more information about AQ agents and using the DBMS_AQADM package

Removing a SYS.AnyData Queue

You use the REMOVE_QUEUE procedure in the DBMS_STREAMS_ADM package to remove an existing SYS.AnyData queue. When you run the REMOVE_QUEUE procedure, it waits until any existing events 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 also can 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 clients that use the queue by setting the `cascade` parameter to `true`. By default, the `cascade` parameter is set to `false`.

For example, to remove a `SYS.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 events from a Streams source queue to a Streams destination queue. This section provides instructions for completing the following tasks:

- Creating a Propagation
- Enabling a Propagation Job
- Scheduling a Propagation Job
- Altering the Schedule of a Propagation Job
- Unscheduling 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

- Disabling a Propagation Job
- Dropping a Propagation

In addition, you can use the features of Oracle Advanced Queuing (AQ) to manage Streams propagations.

> **See Also:** *Oracle Streams Advanced Queuing User's Guide and Reference* for more information about managing propagations with the features of AQ

Creating a Propagation

You can use any of the following procedures to create a propagation:

- `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`
- `DBMS_PROPAGATION_ADM.CREATE_PROPAGATION`

Each of the procedures in the `DBMS_STREAMS_ADM` package creates a propagation with the specified name if it does not already exist, creates either a positive or negative rule set for the propagation if the propagation does not have such a rule set, and may add table, schema, or global rules to the rule set. The `CREATE_PROPAGATION` procedure creates a propagation, but does not create a rule set or rules 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 a SYS.AnyData Queue" on page 10-2 for instructions.
- Create a database link between the database containing the source queue and the database containing the destination queue. See "Configuring Network Connectivity and Database Links" on page 8-11 for information.

Example of Creating a Propagation Using DBMS_STREAMS_ADM

The following is an example that 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.strm01_queue',
    destination_queue_name  => 'strmadmin.strm02_queue@dbs2.net',
    include_dml             => true,
    include_ddl             => true,
    include_tagged_lcr      => false,
    source_database         => 'dbs1.net',
    inclusion_rule          => 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 `strm01_queue` in the current database to `strm02_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 specified by the system.
- 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 specified by the system.
- 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 rules for the propagation.
- Specifies that the source database for the LCRs being propagated is `dbs1.net`, which may or may 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, if one does not exist for the specified database link

See Also:

- "Event Propagation Between Queues" on page 3-5
- "System-Created Rules" on page 6-7
- *Oracle Streams Replication Administrator's Guide* for more information about Streams tags

Example of Creating a Propagation Using DBMS_PROPAGATION_ADM

The following is an example that 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');
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 events 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 events may be captured events or user-enqueued events, or both.
- Specifies that the propagation uses the `dbs2.net` database link to propagate the events

- 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, if one does not exist for the specified database link

See Also:

- "Captured and User-Enqueued Events" on page 3-3
- "Event Propagation Between Queues" on page 3-5

Enabling a Propagation Job

By default, propagation jobs are enabled upon creation. If you disable a propagation job and want to enable it, then use the `ENABLE_PROPAGATION_SCHEDULE` procedure in the `DBMS_AQADM` package.

For example, to enable a propagation job that propagates events from the `strmadmin.strm01_queue` source queue using the `dbs2.net` database link, run the following procedure:

```
BEGIN
  DBMS_AQADM.ENABLE_PROPAGATION_SCHEDULE(
    queue_name  => 'strmadmin.strm01_queue',
    destination => 'dbs2.net');
END;
/
```

Note: Completing this task affects all propagations that propagate events from the source queue to all destination queues that use the `dbs2.net` database link.

See Also:

- *Oracle Streams Advanced Queuing User's Guide and Reference* for more information about using the `ENABLE_PROPAGATION_SCHEDULE` procedure
- "Propagation Jobs" on page 3-17

Scheduling a Propagation Job

You can schedule a propagation job using the SCHEDULE_PROPAGATION procedure in the DBMS_AQADM package. If there is a problem with a propagation job, then unscheduling and scheduling the propagation job may correct the problem.

For example, the following procedure schedules a propagation job that propagates events from the strmadmin.strm01_queue source queue using the dbs2.net database link:

```
BEGIN
  DBMS_AQADM.SCHEDULE_PROPAGATION(
   queue_name  => 'strmadmin.strm01_queue',
   destination => 'dbs2.net');
END;
/
```

Note: Completing this task affects all propagations that propagate events from the source queue to all destination queues that use the dbs2.net database link.

See Also:

- "Unscheduling a Propagation Job" on page 10-13
- *Oracle Streams Advanced Queuing User's Guide and Reference* for more information about using the SCHEDULE_PROPAGATION procedure
- "Propagation Jobs" on page 3-17

Altering the Schedule of a Propagation Job

You can alter the schedule of an existing propagation job using the ALTER_PROPAGATION_SCHEDULE procedure in the DBMS_AQADM package.

For example, suppose you want to alter the schedule of a propagation job that propagates events from the strmadmin.strm01_queue source queue using the dbs2.net database link. The following procedure sets the propagation job to propagate events every 15 minutes (900 seconds), with each propagation lasting 300 seconds, and a 25 second wait before new events in a completely propagated queue are propagated.

```
BEGIN
  DBMS_AQADM.ALTER_PROPAGATION_SCHEDULE(
   queue_name  => 'strmadmin.strm01_queue',
   destination => 'dbs2.net',
   duration    => 300,
   next_time   => 'SYSDATE + 900/86400',
   latency     => 25);
END;
/
```

Note: Completing this task affects all propagations that propagate events from the source queue to all destination queues that use the `dbs2.net` database link.

See Also:

- *Oracle Streams Advanced Queuing User's Guide and Reference* for more information about using the `ALTER_PROPAGATION_SCHEDULE` procedure
- "Propagation Jobs" on page 3-17

Unscheduling a Propagation Job

You can unschedule a propagation job using the `UNSCHEDULE_PROPAGATION` procedure in the `DBMS_AQADM` package. If there is a problem with a propagation job, then unscheduling and scheduling the propagation job may correct the problem.

For example, the following procedure unschedules a propagation job that propagates events from the `strmadmin.strm01_queue` source queue using the `dbs2.net` database link:

```
BEGIN
  DBMS_AQADM.UNSCHEDULE_PROPAGATION(
   queue_name  => 'strmadmin.strm01_queue',
   destination => 'dbs2.net');
END;
/
```

Note: Completing this task affects all propagations that propagate events from the source queue to all destination queues that use the `dbs2.net` database link.

See Also:

- "Scheduling a Propagation Job" on page 10-12
- *Oracle Streams Advanced Queuing User's Guide and Reference* for more information about using the `UNSCHEDULE_PROPAGATION` procedure
- "Propagation Jobs" on page 3-17

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 an event 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 rules 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 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:

- "Event Propagation Between Queues" on page 3-5
- "System-Created Rules" on page 6-7

Adding Rules to the Positive Rule Set for a Propagation

The following is an example that 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.strm01_queue',
    destination_queue_name  => 'strmadmin.strm02_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, and the other rule evaluates to `TRUE` for DDL LCRs that contain DDL changes to the `hr.locations` table. The rule names are specified by the system.
- 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 is an example that 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.strm01_queue',
    destination_queue_name  => 'strmadmin.strm02_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 specified by the system.
- 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 specify that you want to 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.

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.

In addition, if you want 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 6-6

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 events in the source queue to the destination queue.

Disabling a Propagation Job

To stop a propagation job, use the DISABLE_PROPAGATION_SCHEDULE procedure in the DBMS_AQADM package.

For example, to stop a propagation job that propagates events from the strmadmin.strm01_queue source queue using the dbs2.net database link, run the following procedure:

```
BEGIN
  DBMS_AQADM.DISABLE_PROPAGATION_SCHEDULE(
    queue_name  => 'strmadmin.strm01_queue',
    destination => 'dbs2.net');
END;
/
```

Note:

- Completing this task affects all propagations that propagate events from the source queue to all destination queues that use the `dbs2.net` database link.
- The `DISABLE_PROPAGATION_SCHEDULE` disables the propagation job immediately. It does not wait for the current duration to end.

See Also: *Oracle Streams Advanced Queuing User's Guide and Reference* for more information about using the `DISABLE_PROPAGATION_SCHEDULE` procedure

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 sets 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 and negative rule set for the propagation may be dropped. If this procedure drops a rule set, then it also drops any rules 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 queues of type `SYS.AnyData`. These queues stage user messages whose payloads are of `SYS.AnyData` type, and a `SYS.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 a SYS.AnyData Wrapper and Enqueuing Them
- Dequeuing a Payload That Is Wrapped in a SYS.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:

- "SYS.AnyData Queues and User Messages" on page 3-12 for conceptual information about messaging in Streams
- "Configuring a Streams Administrator" on page 8-2
- *Oracle Streams Advanced Queuing User's Guide and Reference* for more information about AQ
- *PL/SQL Packages and Types Reference* for more information about the `SYS.AnyData` type

Wrapping User Message Payloads in a SYS.AnyData Wrapper and Enqueuing Them

You can wrap almost any type of payload in a `SYS.AnyData` payload. The following sections provide examples of enqueuing messages into, and dequeuing messages from, a `SYS.AnyData` queue.

The following steps illustrate how to wrap payloads of various types in a SYS.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 a SYS.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 events.

   ```
   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 `SYS.AnyData` type and enqueues a message containing the payload into an existing `SYS.AnyData` queue.

```
CREATE OR REPLACE PROCEDURE oe.enq_proc (payload SYS.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(SYS.AnyData.ConvertVarchar2('Chemicals - SW'));
COMMIT;
```

   `NUMBER` type:

```
EXEC oe.enq_proc(SYS.AnyData.ConvertNumber('16'));
COMMIT;
```

   User-defined type:

```
BEGIN
  oe.enq_proc(SYS.AnyData.ConvertObject(oe.cust_address_typ(
    '1646 Brazil Blvd','361168','Chennai','Tam', 'IN')));
END;
/
COMMIT;
```

See Also: "Viewing the Contents of User-Enqueued Events in a Queue" on page 14-25 for information about viewing the contents of these enqueued messages

Dequeuing a Payload That Is Wrapped in a SYS.AnyData Payload

The following steps illustrate how to dequeue a payload wrapped in a `SYS.AnyData` payload. This example assumes that you have completed the steps in "Wrapping User Message Payloads in a SYS.AnyData Wrapper and Enqueuing Them" on page 10-20.

To dequeue messages, 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     SYS.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 messages into a SYS.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 events based on rules. In this example, the messaging client uses a rule so that it only dequeues messages that involve the oe.orders table, and 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, and the message can be dequeued by the messaging client because the message satisfies the messaging client's rule sets.

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 who 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;
/
```

   To determine the current mail host, mail port, and send from settings for a database, you can use procedures in the DBMS_AQELM package to get this information. 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 a `SYS.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      => SYS.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            SYS.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 is an example of 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 may 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 14-23
- "Viewing Message Notifications" on page 14-24
- Chapter 6, "How Rules Are Used In Streams" for more information about rule sets for Streams clients and for information about how events 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

11 Managing an Apply Process

A Streams apply process dequeues logical change records (LCRs) and user messages 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 8-2
- *Oracle Streams Replication Administrator's Guide* for more information about managing DML handlers, DDL handlers, and Streams tags for an apply process

Creating, Starting, Stopping, and Dropping an Apply Process

This section contains instructions for creating, starting, stopping and dropping an apply process. It contains the following topics:

- Creating an Apply Process
- Starting an Apply Process
- Stopping an Apply Process
- Dropping 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 or negative rule set for the apply process if the apply process does not have such a rule set, and may add table rules, schema rules, global rules, 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 rules 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 handlers, an apply user, an apply tag, and whether to apply captured or user-enqueued events.

Before you create an apply process, create a `SYS.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 may 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 a SYS.AnyData Queue" on page 10-2
- "Supplemental Logging in a Streams Environment" on page 2-15 for information about when supplemental logging is required
- "Specifying Supplemental Logging at a Source Database" on page 9-33

Examples of Creating an Apply Process Using DBMS_STREAMS_ADM

The first example in this section creates an apply process that applies captured events, and the second example in this section creates an apply process that applies user-enqueued events. A single apply process cannot apply both captured and user-enqueued events.

- Example That Creates an Apply Process for Captured Events Using DBMS_STREAMS_ADM
- Example That Creates an Apply Process for User-Enqueued Events Using DBMS_STREAMS_ADM

See Also:

- "Apply Process Creation" on page 4-17
- "System-Created Rules" on page 6-7
- *Oracle Streams Replication Administrator's Guide* for more information about Streams tags

Example That Creates an Apply Process for Captured Events Using DBMS_STREAMS_ADM

The following is an example that runs the ADD_SCHEMA_RULES procedure in the DBMS_STREAMS_ADM package to create an apply process that applies captured events:

```
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 events 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 specified by the system.
- 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 specified by the system.

- 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 events are dequeued by the apply process. If the apply process dequeues an LCR with a source database that is different than `dbs1.net`, then an error is raised.

Example That Creates an Apply Process for User-Enqueued Events Using DBMS_STREAMS_ADM The following is an example that 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 events 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 specified by the system, and the rule set does not use an evaluation context.

- Creates one rule that evaluates to `TRUE` for user-enqueued events 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 specified by the system.
- 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 6-36
- "Evaluation Contexts for Message Rules" on page 6-48

Examples of Creating an Apply Process Using DBMS_APPLY_ADM

The first example in this section creates an apply process that applies captured events, and the second example in this section creates an apply process that applies user-enqueued events. A single apply process cannot apply both captured and user-enqueued events.

- Example That Creates Apply Process for Captured Events Using DBMS_APPLY_ADM
- Example That Creates an Apply Process for User-Enqueued Events Using DBMS_APPLY_ADM

See Also:

- "Apply Process Creation" on page 4-17
- "Event Processing Options with an Apply Process" on page 4-4 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 events to a non-Oracle database using the `apply_database_link` parameter

Example That Creates Apply Process for Captured Events Using DBMS_APPLY_ADM The following is an example that runs the `CREATE_APPLY` procedure in the `DBMS_APPLY_ADM` package to create an apply process that applies captured events:

```
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 LCRs, and not user-enqueued events. Therefore, if an LCR that was constructed by a user application, not by 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 rules in the apply process rule sets determine which events are dequeued by the apply process. If the apply process dequeues an LCR with a source database that is different than `dbs1.net`, then an error is raised.

Example That Creates an Apply Process for User-Enqueued Events Using DBMS_APPLY_ADM The following is an example that runs the `CREATE_APPLY` procedure in the `DBMS_APPLY_ADM` package to create an apply process that applies user-enqueued events:

```
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 events, and not captured events.
- 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;
/
```

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 sets 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 may be dropped. If this procedure drops a rule set, then it also drops any rules 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 11-32

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 an event if it evaluates to TRUE for at least one rule in the positive rule set and discards an event 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 rules 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 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 6-7

Adding Rules to the Positive Rule Set for an Apply Process

The following is an example that 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 specified by the system.
- Creates one rule that evaluates to `TRUE` for DDL LCRs that contain DDL changes to the `hr.departments` table. The rule name is specified by the system.
- 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 is an example that 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 specified by the system.
- Creates one rule that evaluates to TRUE for DDL LCRs that contain DDL changes to the hr.regions table. The rule name is specified by the system.
- 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 specify that you want to 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.

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 the positive rule set.

In addition, if you want 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 6-6

Removing a Rule Set for an Apply Process

You specify that you want to 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 set 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 events does not have a positive or negative rule set, then the apply process applies all captured events in its queue. Similarly, if an apply process that applies user-enqueued events does not have a positive or negative rule set, then the apply process applies all user-enqueued events 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`, 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 and foreign key columns in the tables for which an apply process applies changes. Supplemental logging may be required for other columns in these tables as well, depending on your configuration.

See Also:

- "Apply Process Parameters" on page 4-19
- The `DBMS_APPLY_ADM.SET_PARAMETER` procedure in the *PL/SQL Packages and Types Reference* for detailed information about the apply process parameters
- "Specifying Supplemental Logging at a Source Database" on page 9-33

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 sets and who runs user-defined apply handlers. 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 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

This section contains instructions for setting and removing the message handler for an apply process.

See Also:

- "Event Processing with an Apply Process" on page 4-3
- *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 `strmadmin` schema.

```
BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name      => 'strm03_apply',
    message_handler => 'strmadmin.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

This section contains instructions for creating, specifying, and 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 may apply captured or user-enqueued events. For a captured row LCR, a commit directive contains the commit SCN of the transaction from the source database, but for a user-enqueued event, 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 save point created in the procedure.

If a precommit handler raises an exception, then the entire apply transaction is rolled back, and all of the events in the transaction are moved to the error queue.

For example, a precommit handler may be used for auditing the row LCRs applied by an apply process. Such a precommit handler is used with one or more separate DML handlers 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 Events Using Precommit Handlers" on page 4-8
- *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 may 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 Event Enqueues by Apply Processes

This section contains instructions for setting a destination queue into which apply processes that use a specified rule in a positive rule set will enqueue events 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 14-47

Setting the Destination Queue for Events That Satisfy a Rule

You use the `SET_ENQUEUE_DESTINATION` procedure in the `DBMS_APPLY_ADM` package to set a destination queue for events that satisfy a certain 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 specified rule's action context to specify the queue. Any apply process in the local database with the `employees5` rule in its positive rule set will enqueue an event into `hr.change_queue` if the event 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 events into the specified queue. If the queue is a secure queue, then the apply user must be a secure queue user of the queue.

An event that has been enqueued into an queue using the `SET_ENQUEUE_DESTINATION` procedure is the same as any other user-enqueued event. Such events 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 event into the destination queue.

See Also:

- "Enabling a User to Perform Operations on a Secure Queue" on page 10-3
- "Enqueue Destinations for Events During Apply" on page 6-52 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 events that satisfy a certain 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, events 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 an event into hr.change_queue if the event satisfies the employees5 rule.

Specifying Execute Directives for Apply Processes

This section contains instructions for setting an apply process execute directive for events 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 14-48

Specifying That Events That Satisfy a Rule Are Not Executed

You use the SET_EXECUTE procedure in the DBMS_APPLY_ADM package to specify that apply processes do not execute events that satisfy a certain 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 an event that satisfies the rule.

For example, to specify that apply processes do not execute events 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 specified rule's action context to specify the execution directive. Any apply process in the local database with the departments8 rule in its positive rule set will not execute an event if the event satisfies the departments8 rule. That is, if the event 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 an event 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 may be used with the `SET_ENQUEUE_DESTINATION` procedure if you want to enqueue events that satisfy a particular rule into a destination queue without executing these events. After an event is enqueued using the `SET_ENQUEUE_DESTINATION` procedure, it is a user-enqueued event 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 Events During Apply" on page 6-52 for more information about how the `SET_EXECUTE` procedure modifies the action context of the specified rule
- "Specifying Event Enqueues by Apply Processes" on page 11-21

Specifying That Events That Satisfy a Rule Are Executed

You use the `SET_EXECUTE` procedure in the `DBMS_APPLY_ADM` package to specify that apply processes execute events that satisfy a certain rule. Specifically, you set the `execute` parameter in this procedure to `true` for the rule. By default, each apply process executes events that satisfy a rule in the positive rule set for the apply process, assuming that the event 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 events 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 an event if the event satisfies the departments8 rule. That is, if the event is an LCR, then an apply process applies the change in the LCR to the relevant database object. Also, an apply process sends an event that satisfies this rule to an apply handler if it is configured to do so.

Managing an Error Handler

This section contains instructions for creating, setting, and removing an error handler.

See Also: "Event Processing with an Apply Process" on page 4-3

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 SYS.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: Certain restrictions 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 restrictions.

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 LCRs 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 may 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 SYS.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 that includes the `regions_pk_error` procedure:

```
CREATE OR REPLACE PACKAGE BODY errors_pkg AS
 PROCEDURE regions_pk_error (
   message            IN SYS.ANYDATA,
   error_stack_depth  IN NUMBER,
   error_numbers      IN DBMS_UTILITY.NUMBER_ARRAY,
   error_messages     IN EMSG_ARRAY )
 IS
  reg_id     NUMBER;
  ad         SYS.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 := Sys.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. However, if you are modifying only one value in the LCR, then the `GET_VALUE` member function and `SET_VALUE` member procedure may 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 either can 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.

You can set the 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

This section contains instructions for retrying and deleting apply errors.

See Also:

- "The Error Queue" on page 4-22
- "Checking for Apply Errors" on page 14-48
- "Displaying Detailed Information About Apply Errors" on page 14-50
- *Oracle Streams Replication Administrator's Guide* for information about the possible causes of apply errors

Retrying Apply Error Transactions

The following sections describe how to retry a specific error transaction and how to retry all error transactions for an apply process. You may need to make DML or DDL changes to database objects to correct the conditions that caused one or more apply errors before you retry apply error transactions. You may also have one or more capture processes configured to capture changes to the same database objects. However, you may not want the changes captured. In this case, you can set the 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

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. 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);
END;
/
```

If `execute_as_user` is `true`, then the apply process reexecutes the transaction in the security context of the current user. If `execute_as_user` is `false`, then the apply process reexecutes 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 handlers. This user must also have dequeue privileges on the queue used by the apply process.

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

The following sections describe how to delete a specific error transaction and how to 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, 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.

12 Managing Rules and Rule-Based Transformations

A Streams environment uses rules to control the behavior of capture processes, propagations, apply processes, and messaging clients. A Streams environment uses rule-based transformations to modify an event that results when a rule evaluates to `TRUE`. Transformations can occur during capture, propagation, apply, or dequeue of an event. In addition, you can create custom applications that are clients of the rules engine. This chapter contains instructions for managing rule sets, rules, and rule-based transformations.

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.

Note: This chapter does not contain examples for creating evaluation contexts, nor does it contain examples for evaluating events using the `DBMS_RULE.EVALUATE` procedure. See Chapter 17, "Rule-Based Application Example" for these examples.

See Also:

- Chapter 5, "Rules"
- Chapter 6, "How Rules Are Used In Streams"
- "Rule-Based Transformations" on page 6-63
- "Configuring a Streams Administrator" on page 8-2

Managing Rule Sets and Rules

You can change a rule or rule set without stopping Streams capture processes, propagations, and apply processes that use the rule or rule set. Streams will detect the change immediately after it is committed. If you need precise control over which events use the new version of a rule or rule set, then you should stop the relevant capture processes and apply processes and disable the relevant propagation jobs, change the rule or rule set, and then restart the stopped processes and propagation jobs.

This section provides instructions for completing the following tasks:

- Creating a Rule Set
- Creating a Rule
- Adding a Rule to a Rule Set
- Altering a Rule
- Modifying System-Created Rules
- Removing a Rule from a Rule Set
- Dropping a Rule
- Dropping a Rule Set

See Also:

- "Stopping a Capture Process" on page 9-26
- "Disabling a Propagation Job" on page 10-18
- "Stopping an Apply Process" on page 11-10

Creating a Rule Set

The following is an example that 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 also can 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 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 9-4
- "Example of Creating a Propagation Using DBMS_STREAMS_ADM" on page 10-9
- "Example That Creates an Apply Process for Captured Events Using DBMS_STREAMS_ADM" on page 11-4

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's name using the rule_name parameter and the rule's 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 also can 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 or the negative rule set for a Streams client. ADD_SUBSET_PROPAGATION_RULES and ADD_SUBSET_RULES can only add rules to the positive rule set for a Streams client.

See Also:

- "Example of Creating a Local Capture Process Using DBMS_STREAMS_ADM" on page 9-4
- "Example of Creating a Propagation Using DBMS_STREAMS_ADM" on page 10-9
- "Example That Creates an Apply Process for Captured Events Using DBMS_STREAMS_ADM" on page 11-4

Creating a Rule With an Action Context

To create a rule with an action context, run the CREATE_RULE procedure and specify the rule's name using the rule_name parameter, the rule's condition using the condition parameter, and the rule's action context using the action_context parameter. You add a name-value pair to a rule's 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', SYS.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 5-10 for a scenario that uses such a name-value pair in an action context

Adding a Rule to a Rule Set

The following is an example that 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.

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's condition
- Change a rule's evaluation context
- Remove a rule's evaluation context
- Modify a name-value pair in a rule's action context
- Add a name-value pair to a rule's action context
- Remove a name-value pair from a rule's 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's 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 12-4. 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 sets that contain the rule.
- If you want to alter a rule but retain the rule's 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's Action Context

To modify a name-value pair in a rule's action context, you first remove the name-value pair from the rule's action context and then add a different name-value pair to the rule's 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's action context and then adding a different name-value pair back to the rule's 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 12-5.

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's 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's action context, this example selects the rule's 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,
                  SYS.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's 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,
                SYS.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 5-10 for a scenario that uses similar name-value pairs in an action context

Removing a Name-Value Pair from a Rule's 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 12-5
- "Modifying a Name-Value Pair in a Rule's Action Context" on page 12-8
- "Adding a Name-Value Pair to a Rule's Action Context" on page 12-10

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 rules are rules created by running a procedure in the DBMS_STREAMS_ADM package. If you want to use a rule-based transformation for a system-created rule, then you can use the SET_RULE_TRANSFORM_FUNCTION procedure in the DBMS_STREAMS_ADM package.

Also, if you cannot create a rule with the 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:

- "Rule-Based Transformations" on page 6-63
- Chapter 14, "Monitoring a Streams Environment" for more information about the data dictionary views related to Streams

Removing a Rule from a Rule Set

The following is an example that 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.

Dropping a Rule

The following is an example that 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 sets. 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.

Dropping a Rule Set

The following is an example that 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 rules, then these rules are not dropped. If the `delete_rules` parameter is set to `true`, then any rules in the rule set, which are not in another rule set, are dropped from the database automatically. If some of the rules in the rule set are in one or more other rule sets, then these rules are not dropped.

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 5-17
- The `GRANT_SYSTEM_PRIVILEGE` and `GRANT_OBJECT_PRIVILEGE` procedures in the `DBMS_RULE_ADM` package in *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 contexts, rule sets, and rules 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.

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.

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 contexts, rule sets, and rules.

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;
/
```

Managing Rule-Based Transformations

In Streams, a rule-based transformation is any modification to an event that results when a rule in a positive rule set evaluates to `TRUE`. You use the `SET_RULE_TRANSFORM_FUNCTION` procedure in the `DBMS_STREAMS_ADM` package to add, alter, or remove a rule-based transformation for a rule. This procedure modifies the rule's action context to specify the rule-based transformation.

This section provides instructions for completing the following tasks:

- Creating a Rule-Based Transformation
- Altering a Rule-Based Transformation
- Removing a Rule-Based Transformation

Attention: Do not modify `LONG`, `LONG RAW` or LOB column data in an LCR. This includes rule-based transformation functions, DML handlers, and error handlers.

Note:

- There is no automatic locking mechanism for a rule's action context. Therefore, make sure an action context is not updated by two or more sessions at the same time.
- When you perform 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 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.
- The 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 clients ignore the rule-based transformation.

See Also: "Rule-Based Transformations" on page 6-63

Creating a Rule-Based Transformation

A function in a rule-based transformation must have the following signature:

```
FUNCTION user_function (
   parameter_name   IN  SYS.AnyData)
RETURN SYS.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 a `SYS.AnyData` encapsulation of an event, and the function must return a `SYS.AnyData` encapsulation of an event.

The following steps outline the general procedure for creating a rule-based transformation:

1. Create a PL/SQL function that performs the transformation.

   Caution: Make sure the transformation function is deterministic. A deterministic function will always return 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 may 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 rule-based transformation for a messaging client may prevent the messaging client from dequeuing events.

   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 may 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 SYS.AnyData)
RETURN SYS.AnyData
IS
  lcr SYS.LCR$_ROW_RECORD;
  rc  NUMBER;
  ob_owner VARCHAR2(30);
  ob_name VARCHAR2(30);
  dep_value_anydata SYS.AnyData;
  dep_value_varchar2 VARCHAR2(30);
BEGIN
  -- Get the type of object
  -- Check if the object type is SYS.LCR$_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',
            SYS.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',
            SYS.ANYDATA.CONVERTVARCHAR2('Management'));
        END IF;
      END IF;
    RETURN SYS.ANYDATA.CONVERTOBJECT(lcr);
    END IF;
  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 rules 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 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 a SYS.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 rule-based transformation using the DBMS_RULE_ADM package. *Oracle Streams Replication Administrator's Guide* contains an example of this.
- 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 rules 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 also can 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 A26

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 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 rule-based 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's action context that specifies the name STREAMS$_TRANSFORM_FUNCTION and a value that is a SYS.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 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
------------ --------------------------- --------------------------
DEPARTMENTS5 STREAMS$_ROW_SUBSET         INSERT
DEPARTMENTS5 STREAMS$_TRANSFORM_FUNCTION hr.executive_to_management
DEPARTMENTS6 STREAMS$_TRANSFORM_FUNCTION hr.executive_to_management
DEPARTMENTS7 STREAMS$_ROW_SUBSET         DELETE
DEPARTMENTS7 STREAMS$_TRANSFORM_FUNCTION hr.executive_to_management
```

You also can view transformation functions using the `DBA_STREAMS_TRANSFORM_FUNCTION` data dictionary view.

> **See Also:** *PL/SQL Packages and Types Reference* for more information about the rule types used in this example

Altering a Rule-Based Transformation

To alter a rule-based transformation, you either can 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 rule-based transformation for rule `DEPARTMENTS5` by changing the transformation function from `hr.execute_to_management` to `hr.executive_to_lead`. This rule based transformation was added to the `DEPARTMENTS5` rule in the example in "Creating a Rule-Based Transformation" on page 12-19.

In Streams, subset rules use name-value pairs in an action context to perform internal transformations that convert `UPDATE` operations into `INSERT` and `DELETE` operations in certain 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 6-27 for more information about row migration

Complete the following steps to alter a 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 A26

   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`. 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.

   Run the following procedure to set the rule-based transformation for rule `DEPARTMENTS5` to `hr.executive_to_lead`:

   ```
   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.

Removing a Rule-Based Transformation

To remove a rule-based transformation from a rule, run the SET_RULE_TRANSFORM_FUNCTION procedure and specify NULL for the transformation function. Specifying NULL removes the name-value pair that specifies the rule-based transformation in the rule's action context. This example removes a rule-based transformation for rule DEPARTMENTS5. This rule-based transformation was added to the DEPARTMENTS5 rule in the example in "Creating a Rule-Based Transformation" on page 12-19.

In Streams, subset rules use name-value pairs in an action context to perform internal transformations that convert UPDATE operations into INSERT and DELETE operations in certain 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 6-27 for more information about row migration

Run the following procedure to remove the 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 removed, you can run the query in Step 1 on page 12-26. 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 6-27 for more information about row migration

13 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 8-2

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 processes, propagations, or apply processes. 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 may 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 queues for propagations from other databases, then disable each propagation job that propagates events to the export database. You can disable a propagation job using the `DISABLE_PROPAGATION_SCHEDULE` procedure in the `DBMS_AQADM` package.
2. Make the necessary changes to your network configuration so that the database links used by the propagation jobs you disabled in Step 1 point to the computer running the import database.

   To complete this step, you may 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 queues.

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 may 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 processes 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 "Monitoring a Streams Capture Process" on page 14-6 for queries that can determine this information.

6. If the export database is not running any apply processes, and is not propagating user-enqueued events, 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 events, 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 clients 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 14-29.

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 15, "Troubleshooting a Streams Environment".

This query 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 to determine the SCN values.

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 13-2.

   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 propagation jobs that are propagating user-enqueued events, then disable these propagation jobs using the `DISABLE_PROPAGATION_SCHEDULE` procedure in the `DBMS_AQADM` package.
10. If the export database is running one or more apply processes, or is propagating user-enqueued events, 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 Downstream Capture" on page 9-6 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 downstream capture processes that were running in the export database in the import 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 9-2 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: *PL/SQL Packages and Types Reference* for detailed information about the actions performed by the `REMOVE_STREAMS_CONFIGURATION` procedure

14 Monitoring a Streams Environment

This chapter provides information about the static data dictionary views and dynamic performance views related to Streams. You can use these views to monitor your Streams environment. This chapter also illustrates example queries that you may want to use to monitor your Streams environment.

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:

- *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

The following table lists the Streams static data dictionary views.

Table 14–1 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
ALL_APPLY_TABLE_COLUMNS	DBA_APPLY_TABLE_COLUMNS	N/A
ALL_APPLY_PARAMETERS	DBA_APPLY_PARAMETERS	N/A
ALL_APPLY_PROGRESS	DBA_APPLY_PROGRESS	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

Table 14–1 Streams Static Data Dictionary Views (Cont.)

ALL_ Views	DBA_ Views	USER_ Views
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
N/A	DBA_STREAMS_ADMINISTRATOR	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
ALL_STREAMS_RULES	DBA_STREAMS_RULES	
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
ALL_STREAMS_UNSUPPORTED	DBA_STREAMS_UNSUPPORTED	N/A

Summary of Streams Dynamic Performance Views

The following list includes the Streams dynamic performance views

- V$BUFFERED_QUEUES
- V$BUFFERED_PUBLISHERS
- V$BUFFERED_SUBSCRIBERS
- V$RULE
- V$RULE_SET
- V$RULE_SET_AGGREGATE_STATS
- V$STREAMS_APPLY_COORDINATOR
- V$STREAMS_APPLY_READER
- V$STREAMS_APPLY_SERVER
- V$STREAMS_CAPTURE

Note: To collect elapsed time statistics in these dynamic performance views, set the `TIMED_STATISTICS` initialization parameter to `true`.

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: *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 only contains 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 may 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 may 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 may 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 Corporation recommends using the REVOKE_ADMIN_PRIVILEGE procedure to revoke privileges from a user. 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 8-2

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 a Streams Capture Process

The following sections contain queries that you can run to display information about a capture process:

- Displaying the Queue, Rule Sets, and Status of Each Capture Process
- Displaying General Information About Each Capture Process
- Displaying Information About Each Downstream Capture Process
- Displaying the Registered Redo Log Files for Each Capture Process
- Displaying the Redo Log Files That Will Never Be Needed by Any Capture Process
- Displaying SCN Values for Each Redo Log File Used by a Capture Process
- Displaying the Last Archived Redo Entry Available to Each Capture Process
- Listing the Parameter Settings for Each Capture Process
- Viewing the Extra Attributes Captured by Each Capture Process
- Determining the Applied SCN for All Capture Processes in a Database
- Determining Redo Log Scanning Latency for Each Capture Process
- Determining Event Enqueuing Latency for Each Capture Process
- Displaying Information About Rule Evaluations for Each Capture Process

See Also:

- Chapter 2, "Streams Capture Process"
- Chapter 9, "Managing a Capture Process"

Displaying the Queue, Rule Sets, and Status of Each Capture Process

You can display the following general 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 may 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 15-2 for an example query that shows the error number and error message if a capture process is aborted

Displaying General Information About Each Capture Process

The query in this section displays the following general information about each capture process in a database:

- The name of the capture process
- The process number (`c`*nnn*)
- The session identifier
- The serial number of the session
- The current state of the capture process, either `INITIALIZING`, `WAITING FOR DICTONARY REDO`, `DICTIONARY INITIALIZATION`, `MINING`, `LOADING`, `CAPTURING CHANGES`, `WAITING FOR REDO`, `EVALUATING RULE`, `CREATING LCR`, `ENQUEUING MESSAGE`, `PAUSED FOR FLOW CONTROL`, or `SHUTTING DOWN`
- The total number of redo entries scanned
- The total number LCRs enqueued

To display this information for each capture process in a database, run the following query:

```
COLUMN CAPTURE_NAME HEADING 'Capture|Name' FORMAT A10
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|Scanned' 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:

```
           Capture         Session                                    Redo    Total
Capture    Process Session  Serial                                 Entries     LCRs
Name       Number       ID  Number State                           Scanned Enqueued
---------- ------- ------- ------- --------------------------- -------- --------
CAPTURE       C001      15       9 CAPTURING CHANGES               14276       51
```

The number of redo entries scanned may be higher than the number of DML and DDL redo entries captured by a capture process. Only DML and DDL redo entries that are captured by a capture process are 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 2-30

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 general 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 may 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 2-17
- "Creating a Downstream Capture Process That Assigns Log Files Implicitly" on page 9-6
- "Is the Capture Process Enabled?" on page 15-2 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 and downstream capture processes.

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 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, while `strm03_capture` is a downstream capture process. The source database for the `strm03_capture` downstream capture process is `dbs1.net`. This query shows the that there are two registered archived redo log files for `strm02_capture` and three registered archived redo log files for `strm02_capture`, and this query shows the name and location of each of these files in the local site's file system.

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 may 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 may be needed.

See Also:

- "The LogMiner Data Dictionary for a Capture Process" on page 2-34 for more information about data dictionary builds
- "Local Capture and Downstream Capture" on page 2-17
- "Creating a Downstream Capture Process That Assigns Log Files Implicitly" on page 9-6
- "ARCHIVELOG Mode and a Capture Process" on page 2-46

Displaying the Redo Log Files That Will Never Be Needed by Any Capture Process

The DBA_LOGMNR_PURGED_LOG data dictionary view lists the redo log files that will never be needed by any capture process at the local database. These redo log files may be removed without affecting any existing capture process at the local database.

To display the redo log files that are no longer needed by any capture process, run the following query:

```
SELECT * FROM DBA_LOGMNR_PURGED_LOG;
```

Your output looks similar to the following:

```
FILE_NAME
--------------------------------------------------------------------
/private1/ARCHIVE_LOGS/1_6_262829418.dbf
```

See Also: "ARCHIVELOG Mode and a Capture Process" on page 2-46

Displaying SCN Values for Each Redo Log File Used by a 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 and downstream capture processes.

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

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 A35
COLUMN FIRST_SCN HEADING 'First SCN' FORMAT 99999999999
COLUMN NEXT_SCN HEADING 'Next SCN' FORMAT 99999999999

SELECT r.CONSUMER_NAME,
       r.NAME,
       r.FIRST_SCN,
       r.NEXT_SCN
  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
--------------- -------------------- ------------ ------------
CAPTURE         /private1/ARCHIVE_LO       202088       202112
                GS/1_3_502628294.dbf

CAPTURE         /private1/ARCHIVE_LO       202112       203389
                GS/1_4_502628294.dbf

CAPTURE         /private1/ARCHIVE_LO       203389       230382
                GS/1_5_502628294.dbf

CAPTURE         /private1/ARCHIVE_LO       230382       235590
                GS/1_6_502628294.dbf

CAPTURE         /private1/ARCHIVE_LO       235590       256147
                GS/1_7_502628294.dbf
```

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 A15
COLUMN PARAMETER HEADING 'Parameter' FORMAT A20
COLUMN VALUE HEADING 'Value' FORMAT A20
COLUMN SET_BY_USER HEADING 'Set by User?' FORMAT A20

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         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 may or may not be set to its default value.

See Also:

- "Capture Process Parameters" on page 2-47
- "Setting a Capture Process Parameter" on page 9-32

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 2-6
- "Managing Extra Attributes in Captured LCRs" on page 9-38
- *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 event dequeued by the relevant apply processes. 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 event scanned by a capture process and the current time. This number may 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 event creation time, which is the time when the data manipulation language (DML) or data definition language (DDL) change generated the redo information for the most recently captured event

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 'Event|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           Event
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 "Event 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 Event Enqueuing Latency for Each Capture Process

You can find the following information about each capture process by running the query in this section:

- The event enqueuing latency, which specifies the number of seconds between when an event was recorded in the redo log and when the event was enqueued by the capture process
- The event creation time, which is the time when the data manipulation language (DML) or data definition language (DDL) change generated the redo information for the most recently enqueued event
- The enqueue time, which is when the capture process enqueued the event into its queue
- The message number of the enqueued event

The information displayed by this query is valid only for an enabled capture process.

Run the following query to determine the event 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 'Event 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 Event 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 `"Event 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 events discarded during prefiltering since the capture process was last started. The capture process determined that these events definitely did not satisfy the capture process rule sets during prefiltering.
- The number of events kept during prefiltering since the capture process was last started. The capture process determined that these events definitely satisfied the capture process rule sets during prefiltering. Such events 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 events after prefiltering since the capture process was last started. These events may or may not satisfy the capture process rule sets. Some of these events may be filtered out after prefiltering without requiring full evaluation while others 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 may 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|Events|Discarded'
  FORMAT 9999999999
COLUMN TOTAL_PREFILTER_KEPT HEADING 'Prefilter|Events|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:

```
                  Prefilter   Prefilter              Undecided
Capture              Events      Events   Prefilter      After        Full
Name              Discarded        Kept Evaluations  Prefilter Evaluations
--------------- ----------- ----------- ----------- ----------- -----------
STREAMS_CAPTURE       68485           0       68570          85          27
```

The total number of prefilter evaluations equals the sum of the prefilter events discarded, the prefilter events kept, and the undecided events.

See Also: "Capture Process Rule Evaluation" on page 2-49

Monitoring a SYS.AnyData Queue and Messaging

The following sections contain queries that you can run to display information about a SYS.AnyData queue:

- Displaying the SYS.AnyData Queues in a Database
- Viewing the Messaging Clients in a Database
- Viewing Message Notifications
- Determining the Consumer of Each User-Enqueued Event in a Queue
- Viewing the Contents of User-Enqueued Events in a Queue

See Also:

- Chapter 3, "Streams Staging and Propagation"
- Chapter 10, "Managing Staging and Propagation"

Displaying the SYS.AnyData Queues in a Database

To display all of the SYS.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 a `SYS.AnyData` queue.

See Also: "Managing SYS.AnyData Queues" on page 10-2

Viewing the Messaging Clients in a Database

You can view the messaging clients 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                                                Positive    Negative
Client                    Queue Owne 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 10-25

Determining the Consumer of Each User-Enqueued Event in a Queue

To determine the consumer for each user-enqueued event 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 events 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 events, not captured events.

See Also: *Oracle Streams Advanced Queuing User's Guide and Reference* for an example that enqueues messages into a `SYS.AnyData` queue

Viewing the Contents of User-Enqueued Events in a Queue

In a `SYS.AnyData` queue, to view the contents of a payload that is encapsulated within a `SYS.AnyData` payload, you query the queue table using the `Access`*`data_type`* static functions of the `SYS.AnyData` type, where *`data_type`* is the type of payload to view.

See Also: "Wrapping User Message Payloads in a SYS.AnyData Wrapper and Enqueuing Them" on page 10-20 for an example that enqueues the events shown in the queries in this section into a `SYS.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 SYS.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 Streams Propagations and Propagation Jobs

The following sections contain queries that you can run to display information about propagations and propagation jobs:

- 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 Events and Bytes Propagated by Each Propagation

See Also:

- Chapter 3, "Streams Staging and Propagation"
- "Managing Streams Propagations and Propagation Jobs" on page 10-7

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. This view contains information about each propagation whose source queue is at the local database.

For example, the following query displays the following information for a propagation named `dbs1_to_dbs2`:

- 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 link used by the propagation

```
COLUMN 'Source Queue' FORMAT A35
COLUMN '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;
```

Your output looks similar to the following:

```
Source Queue                        Destination Queue
----------------------------------- -----------------------------------
STRMADMIN.STREAMS_QUEUE@DBS1.NET    STRMADMIN.STREAMS_QUEUE@DBS2.NET
STRMADMIN.STRM02_QUEUE@DBS1.NET     STRMADMIN.STRM02_QUEUE@DBS2.NET
```

Determining the Rule Sets for Each Propagation

The following query 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 name 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 events before restarting
- 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 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 3-18 for more information about the default propagation schedule for a Streams propagation job
- "Is the Propagation Job Used by a Propagation Enabled?" on page 15-9 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 Events and Bytes Propagated by Each Propagation

All propagation jobs 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 events 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 Events|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 Events    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

Monitoring a Streams Apply Process

The following sections contain queries that you can run to display information about an apply process:

- Determining the Queue, Rule Sets, and Status for Each Apply Process
- Displaying General Information About Each Apply Process
- Listing the Parameter Settings for Each Apply Process
- Displaying Information About Apply Handlers
- Displaying Information About the Reader Server for Each Apply Process
- Determining Capture to Dequeue Latency for an Event
- Displaying General Information About Each Coordinator Process
- Displaying Information About Transactions Received and Applied
- Determining the Capture to Apply Latency for an Event for Each Apply Process
- Displaying Information About the Apply Servers for Each Apply Process
- Displaying Effective Apply Parallelism for an Apply Process
- Viewing Rules That Specify a Destination Queue On Apply
- Viewing Rules That Specify No Execution On Apply
- Checking for Apply Errors
- Displaying Detailed Information About Apply Errors

See Also:

- Chapter 4, "Streams Apply Process"
- Chapter 11, "Managing an Apply Process"

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.

> **See Also:** "Checking for Apply Errors" on page 14-48 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 events applied by the apply process. An apply process may apply either events that were captured by a capture process or events 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 Events Applied' FORMAT A15
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 Events  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 A20
COLUMN VALUE HEADING 'Value' FORMAT A20
COLUMN SET_BY_USER HEADING 'Set by User?' FORMAT A20
```

```
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?
--------------- -------------------- -------------------- --------------------
STRM01_APPLY    COMMIT_SERIALIZATION FULL                 NO
STRM01_APPLY    DISABLE_ON_ERROR     Y                    YES
STRM01_APPLY    DISABLE_ON_LIMIT     N                    NO
STRM01_APPLY    MAXIMUM_SCN          INFINITE             NO
STRM01_APPLY    PARALLELISM          1                    NO
STRM01_APPLY    STARTUP_SECONDS      0                    NO
STRM01_APPLY    TIME_LIMIT           INFINITE             NO
STRM01_APPLY    TRACE_LEVEL          0                    NO
STRM01_APPLY    TRANSACTION_LIMIT    INFINITE             NO
STRM01_APPLY    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 may or may not be set to its default value.

See Also:

- "Apply Process Parameters" on page 4-19
- "Setting an Apply Process Parameter" on page 11-16

Displaying Information About Apply Handlers

This section contains queries that display information about apply process message handlers and error handlers.

See Also:

- "Event Processing with an Apply Process" on page 4-3
- *Oracle Streams Replication Administrator's Guide* for information about monitoring DML handlers and DDL handlers

Displaying All of the Error Handlers for Local Apply

When you specify a local error handler using the SET_DML_HANDLER procedure in the DBMS_APPLY_ADM package at a destination database, you either can specify 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"
```

Because Apply Process Name is NULL for the strmadmin.errors_pkg.regions_pk_error error handler, this handler is a general handler that runs for all of the local apply processes.

See Also: "Managing an Error Handler" on page 11-25

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         "HR"."MES_PROC"
```

See Also: "Managing the Message Handler for an Apply Process" on page 11-17

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|Events|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
                                                       Events
Apply Process Name   Precommit Handler                 Applied
-------------------- ------------------------------ ---------------
STRM01_APPLY         "STRMADMIN"."HISTORY_COMMIT"   Captured
```

See Also: "Managing the Precommit Handler for an Apply Process" on page 11-18

Displaying Information About the Reader Server for Each Apply Process

The reader server for an apply process dequeues events from the queue. The reader server is a parallel execution server that computes dependencies between LCRs and assembles events 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 events dequeued by the reader server, either captured LCRs or user-enqueued messages
- The name of the parallel execution server used by the reader server
- The current state of the reader server, either `IDLE`, `DEQUEUE MESSAGES`, or `SCHEDULE MESSAGES`
- The total number of events 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 A17
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 Events|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 Events
Name              Apply Type                 Name    State                 Dequeued
----------------- ---------------------- ------- ----------------- ------------
APPLY_FROM_MULT2  Captured LCRS              P000    DEQUEUE MESSAGES          3803
APPLY_FROM_MULT1  Captured LCRS              P001    DEQUEUE MESSAGES          2754
```

See Also: "Reader Server States" on page 4-15

Determining Capture to Dequeue Latency for an Event

The query in this section displays the following information about the last event dequeued by each apply process:

- The name of the apply process
- The latency. For captured events, the latency is the amount of time between when the event was created at a source database and when the event was dequeued by the apply process. For user-enqueued events, the latency is the amount of time between when the event enqueued at the local database and when the event was dequeued by the apply process.
- The event creation time. For captured events, the event creation time is the time when the data manipulation language (DML) or data definition language (DDL) change generated the redo information at the source database for the event. For user-enqueued events, the event creation time is the last time the event was enqueued. A user-enqueued event may be enqueued one or more additional times by propagation before it reaches an apply process.
- The time when the event was dequeued by the apply process
- The message number of the event 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 'Event 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 Event Creation    Last Dequeue Time    Message Number
----------------- ------- ----------------- -------------------- --------------
APPLY_FROM_MULT1       36 10:56:51 02/27/03 10:57:27 02/27/03            253962
APPLY_FROM_MULT2       18 13:13:04 02/28/03 13:13:22 02/28/03            633043
```

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 4-16

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 an Event for Each Apply Process

This section contains two different queries that show the capture to apply latency for a particular event. That is, for captured events, these queries show the amount of time between when the event was created at a source database and when the event was applied by the apply process. One query uses the `V$STREAMS_APPLY_COORDINATOR` dynamic performance view, while the other uses the `DBA_APPLY_PROGRESS` static data dictionary view.

Note: These queries assume that the apply process applies captured events, not user-enqueued events.

The following are the major differences between these two queries:

- 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 may show the latency for a more recent transaction than the query on the DBA_APPLY_PROGRESS view.

Both queries display the following information about an event applied by each apply process:

- The apply process name
- The capture to apply latency for the event
- The event creation time. For captured events, the event creation time is the time when the data manipulation language (DML) or data definition language (DDL) change generated the redo information at the source database for the event.
- The time when the event was applied by the apply process
- The message number of the event

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 an event for each apply process:

```
COLUMN APPLY_NAME HEADING 'Apply Process|Name' FORMAT A17
COLUMN 'Latency in Seconds' FORMAT 999999
COLUMN 'Event 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')
        "Event 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 Event Creation    Apply Time         Number
----------------- ------------------ ----------------- ----------------- -------
APPLY_FROM_MULT1                 781 14:05:29 02/28/03 14:18:30 02/28/03  638609
APPLY_FROM_MULT2                 381 13:13:04 02/28/03 13:19:25 02/28/03  633043
```

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 an event for each apply process:

```
COLUMN APPLY_NAME HEADING 'Apply Process|Name' FORMAT A17
COLUMN 'Latency in Seconds' FORMAT 999999
COLUMN 'Event 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')
        "Event 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 Event Creation    Apply Time         Number
----------------- ------------------ ----------------- ----------------- -------
APPLY_FROM_MULT1                 219 14:05:23 02/28/03 14:09:02 02/28/03  638607
APPLY_FROM_MULT2                2641 12:29:21 02/28/03 13:13:22 02/28/03  617393
```

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 messages, the apply servers pass the events 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, either `IDLE`, `RECORD LOW-WATERMARK`, `ADD PARTITION`, `DROP PARTITION`, `EXECUTE TRANSACTION`, `WAIT COMMIT`, `WAIT DEPENDENCY`, `WAIT FOR NEXT CHUNK`, or `TRANSACTION CLEANUP`. See `V$STREAMS_APPLY_SERVER` in the *Oracle Database Reference* for more information about these states.
- The total number of transactions assigned to each apply server since the last time the apply process was started. A transaction may contain more than one event.
- The total number of events 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|Events|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.SERVER_ID;
```

Your output looks similar to the following:

```
                                                                Total     Total
                                                         Transactions    Events
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 4-16

Displaying Effective Apply Parallelism for an Apply Process

In some environments, an apply process may not use all of the apply servers available to it. For example, apply process parallelism may 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 events 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 Events 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 Events 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 an event satisfies the rule, then the apply process enqueues the event 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 Event Enqueues by Apply Processes" on page 11-21
- "Enqueue Destinations for Events During Apply" on page 6-52

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 an event 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 an event satisfies the rule, then the apply process does not execute the event and does not send the event 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 11-23
- "Execution Directives for Events During Apply" on page 6-52

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 'Events 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                                                   Events 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 may 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 4-22
- "Managing Apply Errors" on page 11-32
- *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 LCR events, but you can extend them to display information about any non-LCR events 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 a SYS.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 LCR events.

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.

Step 2 Create a Procedure That Prints the Value in a SYS.AnyData Object

The following procedure prints the value in a specified `SYS.AnyData` object for some selected datatypes. You may add more datatypes to this procedure if you wish.

```
CREATE OR REPLACE PROCEDURE print_any(data IN SYS.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 a SYS.AnyData Object" on page 14-51.

```
CREATE OR REPLACE PROCEDURE print_lcr(lcr IN SYS.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$_ROW_LIST;
  ddl_text  CLOB;
  ext_attr  SYS.AnyData;
BEGIN
  typenm := lcr.GETTYPENAME();
  DBMS_OUTPUT.PUT_LINE('type name: ' || typenm);
  IF (typenm = 'SYS.LCR$_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 14-52.

```
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    SYS.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 14-52.

```
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    SYS.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 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')
```

Monitoring Rules and Rule-Based Transformations

The following sections contain queries that you can run to display information about rules and rule-based transformations:

- Displaying All Rules Used by All Streams Clients
- Displaying the Streams Rules Used by a Specific Streams Client
- Displaying the Current Condition for a Rule
- Displaying Rule Conditions for Streams Rules That Have Been Modified
- Displaying the Evaluation Context for Each Rule Set
- Displaying Information About the Tables Used by an Evaluation Context

- Displaying Information About the Variables Used in an Evaluation Context
- Displaying All of the Rules in a Rule Set
- Displaying the Condition for Each Rule in a Rule Set
- Listing Each Rule that Contains a Specified Pattern in Its Condition
- Displaying Rule-Based Transformations
- Displaying Aggregate Statistics for All Rule Set Evaluations
- Displaying General Information About Rule Set Evaluations
- Determining the Resources Used by Evaluation of Each Rule Set
- Displaying Evaluation Statistics for a Rule

See Also:

- Chapter 5, "Rules"
- Chapter 6, "How Rules Are Used In Streams"
- Chapter 12, "Managing Rules and Rule-Based Transformations"
- "Modifying a Name-Value Pair in a Rule's Action Context" on page 12-8 for information about viewing a rule's action context

Displaying All Rules Used by All Streams Clients

Streams rules are rules created using the `DBMS_STREAMS_ADM` package or the Streams tool in the Oracle Enterprise Manager Console. Streams rules in the rule sets for a Streams client determine the behavior of the Streams client. Streams clients include capture processes, propagations, apply processes, and messaging clients. The rule sets for a Streams client also may 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 and table rules
- 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 rules 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, nor do they display rules created using the DBMS_RULE_ADM package. In addition, 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 rules 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.

See Also: "System-Created Rules" on page 6-7

Determining the Rules in the Positive Rule Set for a Streams Client

The following query displays all of the rules in the positive rule set for an apply process 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 conditions 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 that 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 rule's 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 Rule Conditions for Streams Rules That Have Been Modified" on page 14-64

Determining the Rules in the Negative Rule Set for a Streams Client

The following query displays all of the rules 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 conditions 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 that 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 rule's 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 Rule Conditions for Streams Rules That Have Been Modified" on page 14-64

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 14-60. 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 6-7

Displaying Rule Conditions for Streams Rules That Have Been Modified

It is possible to modify the rule condition of a Streams rule. These modifications may change the behavior of the Streams clients using the Streams rule. In addition, some modifications may 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 only applies 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 6-7

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 5-6
- "Evaluation Contexts Used in Streams" on page 6-45

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 5-6

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 5-6

Displaying All of the Rules in a Rule Set

The query in this section displays the following information about all of the rules 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 6-7

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 conditions. 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 Rule-Based Transformations

A rule-based transformation is any modification to an event that results when a rule in a positive rule set evaluates to `TRUE`. You specify a PL/SQL function that performs the modification.

The following query displays each rule-based transformation specified in a database:

```
COLUMN RULE_OWNER HEADING 'Rule Owner' FORMAT A20
COLUMN RULE_NAME HEADING 'Rule Name' FORMAT A20
COLUMN TRANSFORM_FUNCTION_NAME HEADING 'Transformation Function' FORMAT A30

SELECT RULE_OWNER, RULE_NAME, TRANSFORM_FUNCTION_NAME
  FROM DBA_STREAMS_TRANSFORM_FUNCTION;
```

Your output looks similar to the following:

```
Rule Owner           Rule Name            Transformation Function
-------------------- -------------------- ------------------------------
STRMADMIN            DEPARTMENTS17        hr.executive_to_management
STRMADMIN            DEPARTMENTS18        hr.executive_to_management
STRMADMIN            DEPARTMENTS19        hr.executive_to_management
```

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:

- "Rule-Based Transformations" on page 6-63
- "Managing Rule-Based Transformations" on page 12-18

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 rules
- 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 conditions 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. Therefore, this output shows 1.79 seconds of CPU time and 10.53 seconds of elapsed time.

Displaying General Information About Rule Set Evaluations

You can query the `V$RULE_SET` dynamic performance view to display general information about rule set evaluations 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 rules 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
```

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
```

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_RULES HEADING 'True Evaluations' FORMAT 999999
COLUMN MAYBE_RULES 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';
```

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 sets used by each capture process must filter them out to avoid errors.

For example, if the rule sets for a capture process instruct the capture process to capture all of the changes made to a certain schema, but 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, then 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 only pertains 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 may 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
```

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 processes and apply processes 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 rules that use the GET_COMPATIBLE member function for LCRs to filter out LCRs that are not compatible with particular databases. These rules may be added to the rule sets of capture processes, propagations, 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 6-56 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 14-75 for more information about objects that are not compatible with Streams

Monitoring Streams Performance Using Statspack

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.

15 Troubleshooting a Streams Environment

This chapter contains information about identifying and resolving common problems in a Streams environment.

This chapter contains these topics:

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 Log Files?
- Are You Trying to Configure Downstream Capture Using DBMS_STREAMS_ADM?
- Are More Actions Required for Downstream Capture without a Database Link?

See Also:

- Chapter 2, "Streams Capture Process"
- Chapter 9, "Managing a Capture Process"
- "Monitoring a Streams Capture Process" on page 14-6

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 may 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 9-26
- "Checking the Trace Files and Alert Log for Problems" on page 15-27
- "Streams Capture Processes and Oracle Real Application Clusters" on page 2-27 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 may 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 may 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 14-18
- "Determining Event Enqueuing Latency for Each Capture Process" on page 14-19
- "Capture Process Parallelism" on page 2-48
- "Setting a Capture Process Parameter" on page 9-32

Are Required Redo Log Files Missing?

When a capture process is started or restarted, it may need to scan redo log files that were generated before the log file that contains the start SCN. A capture process must scan these records to keep track of DDL changes to database objects. 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.

See Also:

- "First SCN and Start SCN" on page 2-24
- "Displaying the Registered Redo Log Files for Each Capture Process" on page 14-11

Is a Downstream Capture Process Waiting for Redo Log Files?

If a downstream capture process is not capturing changes, then it may be waiting for redo log files to scan. Redo log files may be registered implicitly or explicitly for a downstream capture process. If redo log files are registered implicitly, then, typically, log transport services transfers the redo log files from the source database to the downstream database. If redo log files are registered explicitly, then you must manually transfer the redo log files to the downstream database and register them with the downstream capture process. In either case, 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 log transport services to transfer redo log files from the source database to the downstream database for this capture process, then make sure log transport services is configured correctly. If log transport services is 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 log transport services to transfer the log files, 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.

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 10g 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 2-17
- Capture Process States on page 2-30
- "Creating a Downstream Capture Process That Assigns Log Files Implicitly" on page 9-6
- "Creating a Downstream Capture Process That Assigns Log Files Explicitly" on page 9-18

Are You Trying to Configure Downstream Capture Using DBMS_STREAMS_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 9-2

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 a Downstream Capture Process That Does Not Use a Database Link" on page 9-13

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 Job Used by a Propagation Enabled?
- Are There Enough Job Queue Processes?
- Is Security Configured Properly for the SYS.AnyData Queue?

See Also:

- Chapter 3, "Streams Staging and Propagation"
- Chapter 10, "Managing Staging and Propagation"
- "Monitoring Streams Propagations and Propagation Jobs" on page 14-27

Does the Propagation Use the Correct Source and Destination Queue?

If events are not appearing in the destination queue for a propagation as expected, then the propagation may not be configured to propagate events 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 may need to remove the existing propagation if it is not appropriate for your environment.

See Also: "Creating a Propagation" on page 10-8

Is the Propagation Job Used by a Propagation Enabled?

For a propagation job to propagate events, the propagation schedule for the propagation job must be enabled. If events are not being propagated by a propagation as expected, then the propagation's propagation job schedule may not be enabled.

You can find the following information about the schedule for a propagation job by running the query in this section:

- The database link used to propagate events from the source queue to the destination queue
- Whether the propagation schedule is enabled or disabled
- The job queue process used to propagate the last event
- The number of consecutive failures when execution of the propagation schedule was attempted. The schedule is disabled automatically if this number reaches 16.
- If there are any propagation errors, then the time of the last error
- If there are any propagation errors, then the error message of the last error

For example, to check whether a propagation job used by a propagation named `dbs1_to_dbs2` is enabled, run the following query:

```
COLUMN DESTINATION_DBLINK HEADING 'Destination|DB Link' FORMAT A15
COLUMN SCHEDULE_DISABLED HEADING 'Schedule' FORMAT A8
COLUMN PROCESS_NAME HEADING 'Process' FORMAT A7
COLUMN FAILURES HEADING 'Number of|Failures' FORMAT 9999
COLUMN LAST_ERROR_TIME HEADING 'Last Error Time' FORMAT A15
COLUMN LAST_ERROR_MSG HEADING 'Last Error Message' FORMAT A18

SELECT p.DESTINATION_DBLINK,
       DECODE(s.SCHEDULE_DISABLED,
                'Y', 'Disabled',
                'N', 'Enabled') SCHEDULE_DISABLED,
       s.PROCESS_NAME,
       s.FAILURES,
       s.LAST_ERROR_TIME,
       s.LAST_ERROR_MSG
  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;
```

If the schedule is enabled currently for the propagation job, then your output looks similar to the following:

```
Destination                       Number of
DB Link         Schedule Process  Failures Last Error Time Last Error Message
--------------- -------- ------- --------- --------------- ------------------
DBS2.NET        Enabled  J001            0
```

If there is a problem, then try the following actions to correct it:

- If a propagation job is disabled, then you can enable it using the ENABLE_PROPAGATION_SCHEDULE procedure in the DBMS_AQADM package, if you have not done so already.
- If the propagation job is disabled, and you do not know why, then check the trace file for the process that last propagated an event. In the previous output, the process is J001.
- If the propagation job is enabled, but is not propagating events, then try unscheduling and scheduling the propagation job.

See Also:

- "Enabling a Propagation Job" on page 10-11
- "Checking the Trace Files and Alert Log for Problems" on page 15-27
- "Unscheduling a Propagation Job" on page 10-13
- "Scheduling a Propagation Job" on page 10-12
- "Displaying the Schedule for a Propagation Job" on page 14-29

Are There Enough Job Queue Processes?

Propagation jobs use job queue processes to propagate events. Make sure the JOB_QUEUE_PROCESSES initialization parameter is set to 2 or higher in each database instance that runs propagations. 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 8-6
- 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 jobs
- *Oracle Database Reference* for more information about the `JOB_QUEUE_PROCESSES` initialization parameter
- *PL/SQL Packages and Types Reference* for more information about job queues

Is Security Configured Properly for the SYS.AnyData Queue?

`SYS.AnyData` queues are secure queues, 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 `SYS.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 may encounter one of the following errors if security is not configured properly for a `SYS.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 3-19

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 10-3 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 a SYS.AnyData Wrapper and Enqueuing Them" on page 10-20 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 Events or User-Enqueued Events?
- Is the Apply Process Queue Receiving the Events to Apply?
- Is a Custom Apply Handler Specified?
- Is the AQ_TM_PROCESSES Initialization Parameter Set to Zero?
- Are There Any Apply Errors in the Error Queue?

See Also:

- Chapter 4, "Streams Apply Process"
- Chapter 11, "Managing an Apply Process"
- "Monitoring a Streams Apply Process" on page 14-32

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 may 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 11-10
- "Displaying Detailed Information About Apply Errors" on page 14-50
- "Checking the Trace Files and Alert Log for Problems" on page 15-27
- "Streams Apply Processes and Oracle Real Application Clusters" on page 4-13 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 cause may 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 may be able to improve performance by adjusting the setting of the `parallelism` apply process parameter.

See Also:

- "Determining the Capture to Apply Latency for an Event for Each Apply Process" on page 14-42
- "Apply Process Parallelism" on page 4-19
- "Setting an Apply Process Parameter" on page 11-16

Does the Apply Process Apply Captured Events or User-Enqueued Events?

An apply process can apply either captured events or user-enqueued events, but not both types of events. If an apply process is not applying events of a certain type, then it may be because the apply process was configured to apply the other type of events. You can check the type of events 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 or user-enqueued events, run the following query:

```
COLUMN APPLY_CAPTURED HEADING 'Type of Events 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 events, then your output looks similar to the following:

```
Type of Events Applied
-------------------------
Captured
```

If an apply process is not applying the expected type of events, then you may need to create a new apply process to apply the events.

See Also:

- "Captured and User-Enqueued Events" on page 3-3
- "Creating a Capture Process" on page 9-2

Is the Apply Process Queue Receiving the Events to Apply?

An apply process must receive events in its queue before it can apply these events. Therefore, if an apply process is applying captured events, then the capture process that captures these events must be enabled, and it must be configured properly. Similarly, if events 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 events may never reach the apply process queue.

The rule sets used by all Streams clients, 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 rules. If the rules for these Streams clients are not configured properly, then the apply process queue may never receive the appropriate events. Also, an event travelling through a stream is the composition of all of the transformations done along the path. For example, if a capture process uses subset rules and performs row migration during capture of an event, and a propagation uses a rule-based transformation on the event to change the table name, then, when the event 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 the problem is caused by the apply process itself:

- If no other destination databases 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 15-2
- "Troubleshooting Propagation Problems" on page 15-7
- "Troubleshooting Problems with Rules and Rule-Based Transformations" on page 15-18

Is a Custom Apply Handler Specified?

You can use PL/SQL procedures to handle events dequeued by an apply process in a customized way. These handlers include DML handlers, DDL handlers, precommit handlers, and message handlers. If an apply process is not behaving as expected, then check the handler procedures used by the apply process, and correct any flaws. You can find the names of these procedures by querying the `DBA_APPLY_DML_HANDLERS` and `DBA_APPLY` data dictionary views. You may need to modify a handler procedure or remove it to correct an apply problem.

See Also:

- "Event Processing Options with an Apply Process" on page 4-4 for general information about apply handlers
- Chapter 11, "Managing an Apply Process" for information about managing apply handlers
- "Displaying Information About Apply Handlers" on page 14-35 for queries that display information about apply handlers

Is the AQ_TM_PROCESSES Initialization Parameter Set to Zero?

If an apply process is not applying events, but there are events that satisfy the apply process rule sets 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 events.

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.

To determine whether there are captured events 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 events in a queue, you can query the queue table for the queue.

See Also:

- "Viewing the Contents of User-Enqueued Events in a Queue" on page 14-25
- *Oracle Streams Replication Administrator's Guide* for information about monitoring buffered queues
- *Oracle Streams Advanced Queuing User's Guide and Reference* for information about the `AQ_TM_PROCESSES` initialization parameter

Are There Any Apply Errors in the Error Queue?

When an apply process cannot apply an event, it moves the event and all of the other events 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 14-48
- "Managing Apply Errors" on page 11-32

Troubleshooting Problems with Rules and Rule-Based Transformations

If a capture process, a propagation, an apply process, or a messaging client is not behaving as expected, then the problem may be that rules or rule-based transformations 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 the Rule-Based Transformations Configured Properly?

See Also:

- Chapter 5, "Rules"
- Chapter 6, "How Rules Are Used In Streams"
- Chapter 12, "Managing Rules and Rule-Based Transformations"

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 may be that the rules in either the positive 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 may be that the rules in the rule sets 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 events that satisfy its rule sets. In general, an event satisfies the rule sets for a Streams client if *no rules* in the negative rule set evaluate to `TRUE` for the event, and *at least one rule* in the positive rule set evaluates to `TRUE` for the event. "Rule Sets and Rule Evaluation of Events" on page 6-4 contains more detailed information about how an event 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:

- "Monitoring Rules and Rule-Based Transformations" on page 14-56
- "Rule Sets and Rule Evaluation of Events" on page 6-4

This section includes the following subsections:

- Checking for Schema and Global Rules
- Checking for Table Rules
- Checking for Subset Rules
- Checking for Message Rules
- Resolving Problems with Rules

Checking for Schema and Global Rules

Schema and global rules in the positive rule set for a Streams client instruct the Streams client to perform its task for all of the events 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 events relating to a particular schema or database, respectively. If a Streams client is not behaving as expected, then it may 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 rules 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. See "Determining the Rules in the Negative Rule Set for a Streams Client" on page 14-62 for an example of a query that shows such rules.

If the query returns any such rules, then the rules returned may 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. See "Determining the Rules in the Positive Rule Set for a Streams Client" on page 14-60 for an example of a query that shows such rules.

Checking for Table Rules

Table rules in the positive rule set for a Streams client instruct the Streams client to perform its task for the events 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 events relating to one or more particular tables.

If a Streams client is not behaving as expected for a particular table, then it may 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 events relating to the table because the table is in a certain database. That is, a global rule in the negative rule set for the Streams client may instruct the Streams client to discard all events from the source database that contains the table, or a global rule in the positive rule set for the Streams client may instruct the Streams client to perform its task for all events 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 events relating to the table because the table is in a certain schema. That is, a schema rule in the negative rule set for the Streams client may instruct the Streams client to discard all events relating to database objects in the schema, or a schema rule in the positive rule set for the Streams client may instruct the Streams client to perform its task for all events relating to database objects in the schema.
- One or more table rules in the rule sets for the Streams client instruct the Streams client to behave in a particular way for events relating to the table.

> **See Also:** "Checking for Schema and Global Rules" on page 15-20

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 may 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. See "Determining the Rules in the Negative Rule Set for a Streams Client" on page 14-62 for an example of a query that shows such rules.

If the query returns any such rules, then the rules returned may be causing the capture process to discard changes to the table. If this 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. See "Determining the Rules in the Positive Rule Set for a Streams Client" on page 14-60 for an example of a query that shows such rules.

> **See Also:** "Table Rules Example" on page 6-20 for more information about specifying table rules

Checking for Subset Rules

A subset rule may 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 should only reside in positive rule sets.

See Also:

- "Table Rules Example" on page 6-20 for more information about specifying subset rules
- "Row Migration and Subset Rules" on page 6-27

Checking for Message Rules

A message rule may be in the rule set used by a propagation, apply process, or messaging client. Message rules only pertain to user-enqueued events of a specific message type, not captured events. A message rule evaluates to TRUE if a user-enqueued event 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 certain user-enqueued events, but the Streams client is not performing its task for these events, then the cause may 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 events. Similarly, if you expect a propagation, apply process, or messaging client to discard certain user-enqueued events, but the Streams client is not discarding these events, then the cause may be that the rules in the positive and negative rule sets for the Streams client do not instruct it to discard these events.

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 may 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 there are any message rules that 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 6-36 for more information about specifying message rules
- "Configuring a Messaging Client and Message Notification" on page 10-25 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 events 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 may 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 12-7
- "Removing a Rule from a Rule Set" on page 12-14
- *Oracle Streams Replication Administrator's Guide* for more information about preparing database objects for instantiation
- "The Streams Data Dictionary" on page 2-45 for more information about the Streams data dictionary

Are the Rule-Based Transformations Configured Properly?

A rule-based transformation is any user-defined modification to an event that results when a rule in a positive rule set evaluates to `TRUE`. A 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 may result.

If a Streams capture process, propagation, apply process, or messaging client is not behaving as expected, then check the rule-based transformation functions specified for 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 may need to modify a transformation function or remove a rule-based transformation to correct the problem. Make sure the name of the function is spelled correctly.

Rule evaluation is done before a 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 the Queue, Rule Sets, and Status of Each Capture Process" on page 14-7 for a query that displays the rule set used by a capture process
- "Displaying Rule-Based Transformations" on page 14-69 for a query that displays the rule-based transformation functions specified for the rules in a rule set
- "Managing Rule-Based Transformations" on page 12-18 for information about modifying or removing rule-based transformations

Checking the Trace Files and Alert Log for Problems

Messages about each capture process, propagation job, 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
- *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 General Information About Each Capture Process" on page 14-8 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 Job Used by a Propagation Enabled?" on page 15-9 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 may 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 may contain 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 14-40 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 14-38 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 14-40 for a query that displays the parallel execution servers used by the apply servers of an apply process

Part III

Example Environments and Applications

This part includes the following detailed examples:

16 Single Database Capture and Apply Example

This chapter illustrates an example of a single database that captures changes to a table, re-enqueues 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

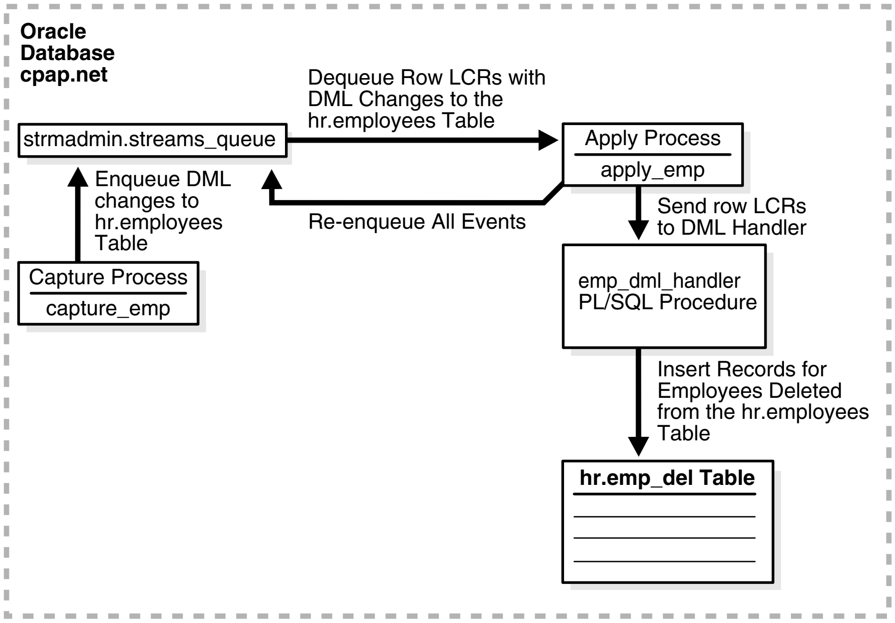
The example in this chapter illustrates using Streams to capture and apply data manipulation language (DML) changes at a single database named `cpap.net`. Specifically, this example captures DML changes to the `employees` table in the `hr` schema, placing row logical change records (LCRs) into a queue named `streams_queue`. Next, an apply process dequeues these row LCRs from the same queue, re-enqueues 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 re-enqueued 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 a `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 if 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 16–1 provides an overview of the environment.

Figure 16–1 Single Database Capture and Apply Example


Oracle Database cpap.net
strmadmin.streams_queue
Dequeue Row LCRs with DML Changes to the hr.employees Table
Apply Process
apply_emp
Enqueue DML changes to hr.employees Table
Re-enqueue All Events
Send row LCRs to DML Handler
Capture Process
capture_emp
emp_dml_handler PL/SQL Procedure
Insert Records for Employees Deleted from the hr.employees Table
hr.emp_del Table

See Also:

- Chapter 2, "Streams Capture Process"
- "LCR Event 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. This parameter specifies the size of the Streams pool. The Streams pool contains captured events. In addition, the Streams pool is used for internal communications during parallel capture and apply. If `STREAMS_POOL_SIZE` is set to zero (the default), then Streams uses the shared pool. In this case, you may need to increase the size of the shared pool.

    See Also: "Setting Initialization Parameters Relevant to Streams" on page 8-6 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.

17 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 processes, propagations, apply processes, or messaging clients are clients of the rules engine in these examples, and no queues are used.

This chapter contains these topics:

- Overview of the Rule-Based Application

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 12, "Managing Rules and Rule-Based Transformations"
- "Monitoring Rules and Rule-Based Transformations" on page 14-56

Overview of the Rule-Based Application

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

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

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

Part IV

Appendixes

This part includes the following appendix:

A XML Schema for LCRs

The XML schema described in this appendix defines the format of a logical change record (LCR).

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
```

This schema definition can be loaded into the database by connecting as `SYS` in SQL*Plus and executing the following file:

```
rdbms/admin/catxlcr.sql
```

The `rdbms` directory is in your Oracle home.

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">
    <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">
    <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 and Maintenance With Streams

This appendix describes performing certain maintenance operations on an Oracle database with little or no down time. These maintenance operations include upgrading to a new version of the Oracle Database, migrating an Oracle Database to a different operating system 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 in the Database Maintenance Process

The following operations typically require substantial database down time:

- Upgrading the version of the database
- Migrating the database to a different operating system
- Migrating the database to a different character set
- Modifying database schema objects to support upgrades to user-created applications
- Applying an Oracle 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 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.

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 for the changes made at the source.
3. Perform the maintenance operation 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.

The upgrade instructions in this appendix assume that all of the following statements are true for the database being upgraded:

- No DML or DDL statements that are not supported by Streams are run during the entire upgrade process.
- 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.
- Job queue processes are not created, modified, or deleted during the upgrade process.
- There are no tables at the database that are master tables for materialized views in other databases.
- Any user-created queues are read-only during the upgrade process.
- 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 sections provide detailed instructions for completing one of the maintenance operations:

- "Performing a Database Version Upgrade Using Streams" on page B-3
- "Performing a Database Maintenance Operation Using Streams" on page B-14

Performing a Database Version Upgrade Using Streams

To use Streams for a database version upgrade, the database must be Oracle9*i* release 2 (9.2). Before you begin the database version upgrade, decide whether you want to use the original Export/Import utilities 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, then you can make the destination database an Oracle Database 10*g* database at the beginning of the operation. Therefore, you do not need to upgrade the destination database after the instantiation.
- If you use RMAN, then the instantiation may be faster than original Export/Import, especially if the database is large, but the database version must be same for RMAN instantiation. Therefore, the destination database is an Oracle9*i* release 2 (9.2) database when it is instantiated. After the instantiation, you must upgrade the destination database.

  Also, Oracle Corporation recommends that you do not use RMAN for instantiation in an environment where distributed transactions are possible. Doing so may cause in-doubt transactions that must be corrected manually.

After you decide which utility you want to use for instantiation, complete the steps in the appropriate section:

- Performing a Database Version Upgrade Using Streams and Original Export/Import
- Performing a Database Version Upgrade Using Streams and RMAN

Performing a Database Version Upgrade Using Streams and Original Export/Import

Complete the following steps to perform a database version upgrade using Export/Import and Oracle Streams:

1. Create an empty Oracle Database 10*g* database. This database will be the destination database during the upgrade process. It may use a different operating system and character set than the source database that is being upgraded.

   See the Oracle installation guide for your operating system if you need to install Oracle, and see *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 upgrade 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 upgrade process.

2. At the source database, make any database objects that were not supported by Streams in Oracle9*i* release 2 (9.2) read-only. In Oracle9*i*, Streams did not support tables with columns of the following datatypes: `NCLOB`, `LONG`, `LONG RAW`, `BFILE`, `ROWID`, and `UROWID`, and user-defined types (including object types, REFs, varrays, and nested tables). In addition, Streams did not support temporary tables, index-organized tables, or object tables. See *Oracle9i Streams* for complete information about unsupported database objects.

3. At the source database, configure a Streams administrator. See *Oracle9i Streams* for instructions. This example assumes 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.

4. While connected as an administrative user in SQL*Plus at the source database, specify database supplemental logging of primary keys or unique indexes (in the absence of primary keys) for all updates. For example:

   ```
   CONNECT SYSTEM/MANAGER@orcl.net

   ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY, UNIQUE INDEX)
   COLUMNS;
   ```

5. While connected as the Streams administrator in SQL*Plus at the source database, create a `SYS.AnyData` queue that will stage changes made to the source database during the upgrade process. For example:

   ```
   CONNECT strmadmin/strmadminpw@orcl.net

   EXEC DBMS_STREAMS_ADM.SET_UP_QUEUE();
   ```

6. While connected as the Streams administrator in SQL*Plus at the source database, configure a local capture process that will capture all supported changes made to the source database and stage these changes in the queue created in Step 5. For example:

   ```
   CONNECT strmadmin/strmadminpw@orcl.net

   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 => true,
      source_database    => NULL);
   END;
   /
   ```

   Do not start the capture process.

7. Instantiate the destination database using original Export/Import by completing the following steps:

   a. At the source database command line, perform a full database export with the `CONSISTENT` export parameter set to `y`:

      ```
      exp SYSTEM/password FULL=y FILE=instant.dmp GRANTS=y ROWS=y
      CONSISTENT=y
      ```

   b. If the source and destination databases are on different computer systems, then transfer the export dump file to the computer system running the destination database.

   c. At the destination database command line in the directory that contains the dump file, perform a full database import with the STREAMS_INSTANTIATION import parameter set to y and the STREAMS_CONFIGURATION import parameters set to n:

      ```
      imp SYSTEM/password FULL=y FILE=instant.dmp COMMIT=y
      LOG=import.log STREAMS_INSTANTIATION=y STREAMS_CONFIGURATION=n
      ```

   See *Oracle Database Utilities* for information about performing an export/import using the original Export and Import utilities.

8. 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.

9. While connected as the Streams administrator in SQL*Plus at the destination database, remove the imported SYS.AnyData queue. For example:

   ```
   CONNECT strmadmin/strmadminpw@stms.net

   BEGIN
     DBMS_STREAMS_ADM.REMOVE_QUEUE(
       queue_name              => 'strmadmin.streams_queue',
       cascade                 => false,
       drop_unused_queue_table => true);
   END;
   /
   ```

10. While connected as the Streams administrator in SQL*Plus at the destination database, re-create the SYS.AnyData queue. This queue will stage changes propagated from the source database. For example:

    ```
    CONNECT strmadmin/strmadminpw@stms.net

    EXEC DBMS_STREAMS_ADM.SET_UP_QUEUE();
    ```

11. Configure your network and Oracle Net so that the source database can communicate with the destination database. See *Oracle Net Services Administrator's Guide* for instructions.

12. While connected as the Streams administrator in SQL*Plus at the source database, 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';
```

13. While connected as the Streams administrator in SQL*Plus at the source database, create a propagation that propagates all changes from the source queue created in Step 5 to the destination queue created in Step 10. For example:

```
CONNECT strmadmin/strmadminpw@orcl.net

BEGIN
  DBMS_STREAMS_ADM.ADD_GLOBAL_PROPAGATION_RULES(
    streams_name            => 'orcl_to_stms',
    source_queue_name       => 'strmadmin.streams_queue',
    destination_queue_name  => 'strmadmin.streams_queue@stms.net',
    include_dml             => true,
    include_ddl             => true,
    include_tagged_lcr      => true,
    source_database         => 'orcl.net');
END;
/
```

14. While connected as the Streams administrator in SQL*Plus at the destination database, create an apply process that applies all changes in the queue created in Step 10. For example:

```
CONNECT strmadmin/strmadminpw@stms.net

BEGIN
  DBMS_STREAMS_ADM.ADD_GLOBAL_RULES(
    streams_type       => 'apply',
    streams_name       => 'apply',
    queue_name         => 'strmadmin.streams_queue',
    include_dml        => true,
    include_ddl        => true,
    include_tagged_lcr => true,
    source_database    => 'orcl.net');
END;
/
```

15. Complete the steps in "Finishing the Database Maintenance Operation" on page B-28.

Performing a Database Version Upgrade Using Streams and RMAN

Complete the following steps to perform a database version upgrade using Recovery Manager (RMAN) and Oracle Streams:

1. Create an empty Oracle9*i* release 2 (9.2) database. This database will be the destination database during the upgrade process. It may use a different operating system and character set than the source database that is being upgraded. Both the source database that is being upgraded and the destination database must be Oracle9*i* release 2 (9.2) databases when you start the upgrade process.

   See the Oracle installation guide for your operating system if you need to install Oracle, and see *Oracle9i 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 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.

2. At the source database, make any database objects that were not supported by Streams in Oracle9*i* release 2 (9.2) read-only. In Oracle9*i* release 2 (9.2), Streams did not support tables with columns of the following datatypes: `NCLOB`, `LONG`, `LONG RAW`, `BFILE`, `ROWID`, and `UROWID`, and user-defined types (including object types, REFs, varrays, and nested tables). In addition, Streams did not support temporary tables, index-organized tables, or object tables. See *Oracle9i Streams* for complete information about unsupported database objects.

3. At the source database, configure a Streams administrator. See *Oracle9i Streams* for instructions. This example assumes that the name of the Streams administrator at the source database is `strmadmin` and that the global name of the source database is `orcl.net`.

4. While connected as an administrative user in SQL*Plus at the source database, specify database supplemental logging of primary keys and unique indexes (in the absence of primary keys) for all updates. For example:

```
CONNECT SYSTEM/MANAGER@orcl.net
```

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY, UNIQUE INDEX)
COLUMNS;
```

5. While connected as the Streams administrator in SQL*Plus at the source database, create a `SYS.AnyData` queue that will stage changes made to the source database during the upgrade process. For example:

```
CONNECT strmadmin/strmadminpw@orcl.net

EXEC DBMS_STREAMS_ADM.SET_UP_QUEUE();
```

6. While connected as the Streams administrator in SQL*Plus at the source database, configure a local capture process that will capture all supported changes made to the source database and stage these changes in the queue created in Step 5. For example:

```
CONNECT strmadmin/strmadminpw@orcl.net

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 => true,
   source_database    => NULL);
END;
/
```

Do not start the capture process.

7. Instantiate the destination database using RMAN `DUPLICATE` command by completing the following steps. These steps provide a general outline for using RMAN to duplicate a database. See the *Oracle9i Recovery Manager User's Guide* for detailed information about using RMAN.

   a. 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.

   b. While connected as an administrative user in SQL*Plus at 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 e.

c. While connected as an administrative user in SQL*Plus at the source database, archive the current online redo log. For example:

```
CONNECT SYSTEM/MANAGER@orcl.net

ALTER SYSTEM ARCHIVE LOG CURRENT;
```

d. Prepare your environment for database duplication, which includes preparing the destination database as an auxiliary instance for duplication. See the "Duplicating a Database with Recovery Manager" chapter in the *Oracle9i Recovery Manager User's Guide* for instructions.

e. Use the RMAN `DUPLICATE` command to instantiate the source database at the destination database. You can use the `UNTIL SCN` clause to specify an SCN for the duplication. Use the until SCN determined in Step b for this clause. Archived redo logs must be available for the until SCN specified and for higher SCN values. Therefore, Step c 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 is an example of 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 439882;
        ALLOCATE AUXILIARY CHANNEL updb DEVICE TYPE sbt;
        DUPLICATE TARGET DATABASE TO updb
        NOFILENAMECHECK;
      }
```

f. While connected as an administrative user in SQL*Plus at the destination database, use the `ALTER SYSTEM` statement to disable the `RESTRICTED SESSION`:

```
CONNECT SYSTEM/MANAGER

ALTER SYSTEM DISABLE RESTRICTED SESSION;
```

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. While connected as an administrative user in SQL*Plus at 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. For example:

```
CONNECT SYSTEM/MANAGER

ALTER DATABASE RENAME GLOBAL_NAME TO stms.net;
```

10. Configure your network and Oracle Net so that the source database and the destination database can communicate with each other. See *Oracle Net Services Administrator's Guide* for instructions.
11. Upgrade the destination database to Oracle Database 10*g*. See the *Oracle Database Upgrade Guide* for instructions.
12. At the destination database, connect as an administrator with `SYSDBA` privilege and run the following procedure:

Attention: Make sure you are connected to the destination database, not the source database, when you run this procedure because it removes the local Streams configuration.

```
CONNECT SYS/CHANGE_ON_INSTALL@stms.net AS SYSDBA

EXEC DBMS_STREAMS_ADM.REMOVE_STREAMS_CONFIGURATION();
```

See *PL/SQL Packages and Types Reference* for more information about the `REMOVE_STREAMS_CONFIGURATION` procedure.

13. While connected as the Streams administrator in SQL*Plus at the destination database, 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';
```

14. While connected as the Streams administrator in SQL*Plus at 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 7e. 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@stms.net

BEGIN
  DBMS_APPLY_ADM.SET_GLOBAL_INSTANTIATION_SCN(
    source_database_name => 'orcl.net',
    instantiation_scn    => 439881,
    recursive            => true);
END;
/
```

15. While connected as the Streams administrator in SQL*Plus at the destination database, remove the imported `SYS.AnyData` queue. For example:

```
CONNECT strmadmin/strmadminpw@stms.net

BEGIN
  DBMS_STREAMS_ADM.REMOVE_QUEUE(
    queue_name             => 'strmadmin.streams_queue',
    cascade                => false,
    drop_unused_queue_table => true);
END;
/
```

16. While connected as the Streams administrator in SQL*Plus at the destination database, re-create the `SYS.AnyData` queue. This queue will stage changes propagated from the source database. For example:

    ```
    CONNECT strmadmin/strmadminpw@stms.net

    EXEC DBMS_STREAMS_ADM.SET_UP_QUEUE();
    ```

17. While connected as the Streams administrator in SQL*Plus at the source database, 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';
    ```

18. While connected as the Streams administrator in SQL*Plus at the source database, create a propagation that propagates all changes from the source queue created in Step 5 to the destination queue created in Step 10. For example:

    ```
    CONNECT strmadmin/strmadminpw@orcl.net

    BEGIN
      DBMS_STREAMS_ADM.ADD_GLOBAL_PROPAGATION_RULES(
        streams_name            => 'orcl_to_stms',
        source_queue_name       => 'strmadmin.streams_queue',
        destination_queue_name  => 'strmadmin.streams_queue@stms.net',
        include_dml             => true,
        include_ddl             => true,
        include_tagged_lcr      => true,
        source_database         => 'orcl.net');
    END;
    /
    ```

19. While connected as the Streams administrator in SQL*Plus at the destination database, create an apply process that applies all changes in the queue created in Step 10. For example:

    ```
    CONNECT strmadmin/strmadminpw@stms.net
    ```

```
BEGIN
  DBMS_STREAMS_ADM.ADD_GLOBAL_RULES(
    streams_type       => 'apply',
    streams_name       => 'apply',
    queue_name         => 'strmadmin.streams_queue',
    include_dml        => true,
    include_ddl        => true,
    include_tagged_lcr => true,
    source_database    => 'orcl.net');
END;
/
```

20. Complete the steps in "Finishing the Database Maintenance Operation" on page B-28.

Performing a Database Maintenance Operation Using Streams

This section describes performing one of the following database maintenance operations on an Oracle Database 10*g* database:

- Migrating the database to a different operating system
- Migrating the database to a different character set
- Modifying database schema objects to support upgrades to user-created applications
- Applying an Oracle software patch

You can use Streams to achieve little or no downtime during one of these operations. During the operation, the source database is the existing database on which you are performing database maintenance. The destination database is the database that will replace the source database at the end of the operation.

Preparing for Upgrades to User-Created Applications

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 DML handler is a user procedure that processes row LCRs resulting from DML statements at a source database. A Streams apply process can pass row LCRs to a DML handler, and the DML handler can modify the row LCR to account for differences between a source database and a destination database.

The process for upgrading your user-created applications using Streams involves modifying and creating the schema objects at the destination database after instantiation. You can use one or more DML handlers at the destination database to process changes from the source database so that they apply to the modified schema objects correctly.

Before you begin the database maintenance operation, you should complete the following tasks to prepare your DML handlers:

- Learn about DML handlers. See "Event Processing Options with an Apply Process" on page 4-4.
- Determine the 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 DML handlers during the database maintenance operation. See *Oracle Streams Replication Administrator's Guide* for information about creating the PL/SQL procedures.

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 operating system, then you must use Export/Import. The RMAN `DUPLICATE` command used for instantiation does not support migrating a database to a different operating system.
- If you are migrating the database to a character set, then you must use Export/Import. The RMAN `DUPLICATE` command used for instantiation does not support migrating a database to a different character set.
- If RMAN is supported for the operation, then using RMAN for the instantiation may be faster than using Export/Import, especially if the database is large.
- Oracle Corporation recommends that you do not use RMAN for instantiation in an environment where distributed transactions are possible. Doing so may cause in-doubt transactions that must be corrected manually.

After you decide which utility you want to use for instantiation, complete the steps in the appropriate section:

- "Performing the Maintenance Operation Using Export/Import and Streams" on page B-16
- "Performing the Maintenance Operation Using RMAN and Streams" on page B-20

Note: The instructions in these sections assume that both the source database and the destination database are running Oracle Database 10*g*.

Performing the Maintenance Operation Using Export/Import and Streams

You may use Data Pump Export/Import or original Export/Import to instantiate the database during the database maintenance operation. Oracle Corporation recommends using Data Pump, and Data Pump may perform the instantiation faster than original Export/Import.

Complete the following steps to perform a maintenance operation using Export/Import and Oracle Streams:

1. Create an empty Oracle Database 10*g* database. This database will be the destination database during the maintenance operation. If you are migrating the database to a different operating system, then create the database on a computer system running this operating system. If you are migrating the database to a different character set, then create a database that uses the character set.

   See the Oracle installation guide for your operating system if you need to install Oracle, and see *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. At the source database, make any database objects that were not supported by Streams in Oracle Database 10*g* read-only. See "Datatypes Captured" on page 2-8 and "Types of Changes Captured" on page 2-10 for information about unsupported objects.

3. At the source database, configure a Streams administrator. See "Configuring a Streams Administrator" on page 8-2 for instructions. This example assumes 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.

4. While connected as an administrative user in SQL*Plus at the source database, specify supplemental logging at the source database of primary keys or unique indexes (in the absence of primary keys) for all updates:

```
CONNECT SYSTEM/MANAGER@orcl.net

ALTER DATABASE ADD SUPPLEMENTAL LOG DATA
   (PRIMARY KEY, UNIQUE, FOREIGN KEY) COLUMNS;
```

5. If you are upgrading user-created applications, then supplementally log any columns at the source database that will be involved in a DML handler at the destination database. These columns must be unconditionally logged at the source database. See *Oracle Streams Replication Administrator's Guide* for information about specifying unconditional supplemental log groups for these columns.

6. While connected as the Streams administrator in SQL*Plus at the source database, create a `SYS.AnyData` queue that will stage changes made to the source database during the maintenance operation. For example:

```
CONNECT strmadmin/strmadminpw@orcl.net

EXEC DBMS_STREAMS_ADM.SET_UP_QUEUE();
```

7. While connected as the Streams administrator in SQL*Plus at the source database, configure a local capture process that will capture all supported changes made to the source database and stage these changes in the queue created in Step 6. For example:

```
CONNECT strmadmin/strmadminpw@orcl.net
```

```
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 => true,
   source_database    => NULL);
END;
/
```

Do not start the capture process.

8. Instantiate the destination database using Oracle Data Pump or original Export/Import. See *Oracle Streams Replication Administrator's Guide* for instructions. In either case, make sure the following parameters are set to the appropriate values:
   - Set the `CONSISTENT` export parameter to `y`.
   - Set the `STREAMS_CONFIGURATION` import parameter to `n`.
   - If you use original Export/Import, then set the `STREAMS_INSTANTIATION` import parameter to `y`. This parameter does not apply to Data Pump imports.
9. 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.
10. If you are applying a patch, then apply the patch now. Follow the instructions included with the patch.
11. While connected as the Streams administrator in SQL*Plus at the destination database, remove the imported `SYS.AnyData` queue. For example:

```
CONNECT strmadmin/strmadminpw@stms.net

BEGIN
  DBMS_STREAMS_ADM.REMOVE_QUEUE(
    queue_name             => 'strmadmin.streams_queue',
    cascade                => false,
    drop_unused_queue_table => true);
END;
/
```

12. While connected as the Streams administrator in SQL*Plus at the destination database, re-create the `SYS.AnyData` queue. This queue will stage changes propagated from the source database. For example:

```
CONNECT strmadmin/strmadminpw@stms.net

EXEC DBMS_STREAMS_ADM.SET_UP_QUEUE();
```

13. Configure your network and Oracle Net so that the source database can communicate with the destination database. See *Oracle Net Services Administrator's Guide* for instructions.

14. While connected as the Streams administrator in SQL*Plus at the source database, 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';
```

15. While connected as the Streams administrator in SQL*Plus at the source database, create a propagation that propagates all changes from the source queue created in Step 5 to the destination queue created in Step 10. For example:

```
CONNECT strmadmin/strmadminpw@orcl.net

BEGIN
  DBMS_STREAMS_ADM.ADD_GLOBAL_PROPAGATION_RULES(
    streams_name            => 'orcl_to_stms',
    source_queue_name       => 'strmadmin.streams_queue',
    destination_queue_name  => 'strmadmin.streams_queue@stms.net',
    include_dml             => true,
    include_ddl             => true,
    include_tagged_lcr      => true,
    source_database         => 'orcl.net');
END;
/
```

16. While connected as the Streams administrator in SQL*Plus at the destination database, create an apply process that applies all changes in the queue created in Step 12. For example:

```
CONNECT strmadmin/strmadminpw@stms.net
```

```
BEGIN
  DBMS_STREAMS_ADM.ADD_GLOBAL_RULES(
    streams_type       => 'apply',
    streams_name       => 'apply',
    queue_name         => 'strmadmin.streams_queue',
    include_dml        => true,
    include_ddl        => true,
    include_tagged_lcr => true,
    source_database    => 'orcl.net');
END;
/
```

17. If you are upgrading user-created applications, then, at the destination database, complete the following steps:

    a. Modify the schema objects in the database to support the upgraded user-applications.

    b. Configure one or more DML handlers 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 DML handler should rename the column in a row LCR that involves the column.

       See *Oracle Streams Replication Administrator's Guide* for information about configuring DML handlers.

18. Complete the steps in "Finishing the Database Maintenance Operation" on page B-28.

Performing the Maintenance Operation Using RMAN and Streams

You may use RMAN to instantiate the database during either of the following database maintenance operations:

- Modifying database schema objects to support upgrades to user-created applications
- Applying an Oracle software patch

However, if you are migrating the database to a different operating system or character set, then you must use Export/Import for instantiation.

See Also: "Performing the Maintenance Operation Using Export/Import and Streams" on page B-16 if you are migrating the database to a different operating system or character set

Complete the following steps to perform a database migration or apply a patch using RMAN and Oracle Streams:

1. Create an empty Oracle Database 10*g* database. This database will be the destination database during the database maintenance operation. Both the source database and the destination database must be Oracle Database 10*g* databases when you start the database maintenance operation.

   See the Oracle installation guide for your operating system if you need to install Oracle, and see *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. At the source database, make any database objects that were not supported by Streams in Oracle Database 10*g* read-only. See "Datatypes Captured" on page 2-8 and "Types of Changes Captured" on page 2-10 for information about unsupported objects.

3. At the source database, configure a Streams administrator. See "Configuring a Streams Administrator" on page 8-2 for instructions. This example assumes 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.

4. While connected as an administrative user in SQL*Plus at the source database, specify supplemental logging at the source database of primary keys or unique indexes (in the absence of primary keys) for all updates:

   ```
   CONNECT SYSTEM/MANAGER@orcl.net

   ALTER DATABASE ADD SUPPLEMENTAL LOG DATA
      (PRIMARY KEY, UNIQUE, FOREIGN KEY) COLUMNS;
   ```

5. If you are upgrading user-created applications, then supplementally log any columns at the source database that will be involved in a DML handler at the destination database. These columns must be unconditionally logged at the source database. See *Oracle Streams Replication Administrator's Guide* for information about specifying unconditional supplemental log groups for these columns.

6. While connected as the Streams administrator in SQL*Plus at the source database, create a `SYS.AnyData` queue that will stage changes made to the source database during the database maintenance operation. For example:

   ```
   CONNECT strmadmin/strmadminpw@orcl.net

   EXEC DBMS_STREAMS_ADM.SET_UP_QUEUE();
   ```

7. While connected as the Streams administrator in SQL*Plus at the source database, configure a local capture process that will capture all supported changes made to the source database and stage these changes in the queue created in Step 6. For example:

   ```
   CONNECT strmadmin/strmadminpw@orcl.net

   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 => true,
      source_database    => NULL);
   END;
   /
   ```

   Do not start the capture process.

8. Instantiate the destination database using RMAN `DUPLICATE` command by completing the following steps. These steps provide a general outline for using RMAN to duplicate a database. See the *Oracle Database Backup and Recovery Advanced User's Guide* for detailed information about using RMAN.

   a. 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.

   b. While connected as an administrative user in SQL*Plus at 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 `748045`. You will set the `UNTIL SCN` option to this value when you use RMAN to duplicate the database in Step e.

c. While connected as an administrative user in SQL*Plus at the source database, archive the current online redo log. For example:

```
CONNECT SYSTEM/MANAGER@orcl.net

ALTER SYSTEM ARCHIVE LOG CURRENT;
```

d. 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.

e. 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 b for this clause. Archived redo logs must be available for the until SCN specified and for higher SCN values. Therefore, Step c 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 is an example of 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 748045;
        ALLOCATE AUXILIARY CHANNEL mgdb DEVICE TYPE sbt;
        DUPLICATE TARGET DATABASE TO mgdb
        NOFILENAMECHECK
        OPEN RESTRICTED;
      }
```

f. While connected as an administrative user in SQL*Plus at the destination database, use the `ALTER SYSTEM` statement to disable the `RESTRICTED SESSION`:

```
CONNECT SYSTEM/MANAGER

ALTER SYSTEM DISABLE RESTRICTED SESSION;
```

9. At the destination database, connect as an administrator with `SYSDBA` privilege and run the following procedure:

Attention: Make sure you are connected to the destination database, not the source database, when you run this procedure because it removes the local Streams configuration.

```
CONNECT SYS/CHANGE_ON_INSTALL AS SYSDBA

EXEC DBMS_STREAMS_ADM.REMOVE_STREAMS_CONFIGURATION();
```

See *PL/SQL Packages and Types Reference* for more information about the `REMOVE_STREAMS_CONFIGURATION` procedure.

10. 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.

11. While connected as an administrative user in SQL*Plus at 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. For example:

```
CONNECT SYSTEM/MANAGER

ALTER DATABASE RENAME GLOBAL_NAME TO stms.net;
```

12. Configure your network and Oracle Net so that the source database and destination databases can communicate with each other. See *Oracle Net Services Administrator's Guide* for instructions.
13. If you are applying a patch, then apply the patch now to the destination database. Follow the instructions included with the patch.
14. While connected as the Streams administrator in SQL*Plus at the destination database, 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';
```

15. While connected as the Streams administrator in SQL*Plus at the destination database, set the instantiation SCN for the entire database and all of the database objects to the until SCN value determined in Step 8b. For example, if the until SCN value is `748045`, then run the following procedure:
16. While connected as the Streams administrator in SQL*Plus at 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 7e. In this example, the until SCN was set to `748045`. Therefore, the instantiation SCN should be set to `748045 - 1`, or `748044`.

```
CONNECT strmadmin/strmadminpw@stms.net

BEGIN
  DBMS_APPLY_ADM.SET_GLOBAL_INSTANTIATION_SCN(
    source_database_name => 'orcl.net',
    instantiation_scn    => 748044,
    recursive            => true);
END;
/
```

17. While connected as the Streams administrator in SQL*Plus at the destination database, remove the imported `SYS.AnyData` queue. For example:

```
CONNECT strmadmin/strmadminpw@stms.net

BEGIN
  DBMS_STREAMS_ADM.REMOVE_QUEUE(
    queue_name              => 'strmadmin.streams_queue',
    cascade                 => false,
    drop_unused_queue_table => true);
END;
/
```

18. While connected as the Streams administrator in SQL*Plus at the destination database, re-create the `SYS.AnyData` queue. This queue will stage changes propagated from the source database. For example:

```
CONNECT strmadmin/strmadminpw@stms.net

EXEC DBMS_STREAMS_ADM.SET_UP_QUEUE();
```

19. While connected as the Streams administrator in SQL*Plus at the source database, 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';
```

20. While connected as the Streams administrator in SQL*Plus at the source database, create a propagation that propagates all changes from the source queue created in Step 6 to the destination queue created in Step 18. For example:

```
CONNECT strmadmin/strmadminpw@orcl.net

BEGIN
  DBMS_STREAMS_ADM.ADD_GLOBAL_PROPAGATION_RULES(
    streams_name            => 'orcl_to_stms',
    source_queue_name       => 'strmadmin.streams_queue',
    destination_queue_name  => 'strmadmin.streams_queue@stms.net',
    include_dml             => true,
    include_ddl             => true,
    include_tagged_lcr      => true,
    source_database         => 'orcl.net');
END;
/
```

21. While connected as the Streams administrator in SQL*Plus at the destination database, create an apply process that applies all changes in the queue created in Step 18. For example:

```
CONNECT strmadmin/strmadminpw@stms.net

BEGIN
  DBMS_STREAMS_ADM.ADD_GLOBAL_RULES(
    streams_type       => 'apply',
    streams_name       => 'apply',
    queue_name         => 'strmadmin.streams_queue',
    include_dml        => true,
    include_ddl        => true,
    include_tagged_lcr => true,
    source_database    => 'orcl.net');
END;
/
```

22. If you are upgrading user-created applications, then, at the destination database, complete the following steps:

    a. Modify the schema objects in the database to support the upgraded user-applications.

    b. Configure one or more DML handlers that modify row LCRs from the source database so that the apply process applies these row LCRs to the modified schema objects correctly. Row LCRs contain information about row changes that result from DML statements. For example, if a column name was changed to support the upgraded user-created applications, then a DML handler should rename the column in a row LCR that involves the column.

       See *Oracle Streams Replication Administrator's Guide* for information about configuring DML handlers.

23. Complete the steps in "Finishing the Database Maintenance Operation" on page B-28.

Finishing the Database Maintenance Operation

Complete the following steps to finish the database maintenance operation:

1. While connected as the Streams administrator in SQL*Plus at the destination database, start the apply process. For example:

```
CONNECT strmadmin/strmadminpw@stms.net

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

2. While connected as the Streams administrator in SQL*Plus at the source database, start the capture process. For example:

```
CONNECT strmadmin/strmadminpw@orcl.net

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

   This step begins the process of replicating changes that were made to the source database during instantiation of the destination database.

3. 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`, and the name of the apply process is `apply`, then run the following query at the source database:

```
CONNECT strmadmin/strmadminpw@orcl.net

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'
    AND APPLY_NAME   = 'APPLY';
```

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 may never be equal because the source database still allows changes.

If this query returns no results, then make sure the Streams clients 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 14-29.

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 15, "Troubleshooting a Streams Environment".

4. While connected as the Streams administrator in SQL*Plus at 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 11-32 for instructions.

5. Disconnect all applications and users from the source database.

6. While connected as an administrative user in SQL*Plus at the source database, restrict access to the database. For example:

```
CONNECT SYSTEM/MANAGER@orcl.net

ALTER SYSTEM ENABLE RESTRICTED SESSION;
```

7. While connected as an administrative user in SQL*Plus at the source database, 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 move on to the next step.

8. While connected as the Streams administrator in SQL*Plus at the destination database, 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 11-32 for instructions.

9. Shut down the source database.
10. At the destination database, remove the Streams components that are no longer needed, including the `SYS.AnyData` queue, the apply process, supplemental logging specifications, and the Streams administrator. See the following sections for instructions:

    - "Removing a SYS.AnyData Queue" on page 10-6
    - "Dropping an Apply Process" on page 11-10

    If you no longer need database supplemental logging, then connect as an administrative user in SQL*Plus at the destination database, and run the following statement to drop it:

    ```
    CONNECT SYSTEM/MANAGER@stms.net

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

    Also, the following statement drops a user named `strmadmin`:

    ```
    DROP USER strmadmin CASCADE;
    ```

11. While connected as an administrative user in SQL*Plus at the destination database, change the global name of the database to match the source database. For example:

    ```
    CONNECT SYSTEM/MANAGER@stms.net

    ALTER DATABASE RENAME GLOBAL_NAME TO orcl.net;
    ```

12. At the destination database, enable any jobs that you disabled earlier.
13. 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 Net Services Administrator's Guide* for instructions.

Index