

Oracle9i

Streams

Release 2 (9.2)

October 2002

Part No. A96571-02

Oracle9i Streams, Release 2 (9.2)

Part No. A96571-02

Copyright © 2002 Oracle Corporation. All rights reserved.

Primary Author: Randy Urbano

Graphic Artist: Valarie Moore

Contributors: Nimar Arora, Lance Ashdown, Ram Avudaiappan, Sukanya Balaraman, Neerja Bhatt, Ragamayi Bhyravabhotla, Diego Cassinera, Debu Chatterjee, Alan Downing, Lisa Eldridge, Curt Elsbernd, Yong Feng, Jairaj Galagali, Brajesh Goyal, Sanjay Kaluskar, Lewis Kaplan, Anand Lakshminath, Jing Liu, Edwina Lu, Raghu Mani, Pat McElroy, Krishnan Meiyappan, 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, Bob Thome, Ramkumar Venkatesan, Wei Wang, Lik Wong, David Zhang

The Programs (which include both the software and documentation) contain proprietary information of Oracle Corporation; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent and other intellectual and industrial property laws. Reverse engineering, disassembly or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. Oracle Corporation does not warrant that this document is error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Oracle Corporation.

If the Programs are delivered to the U.S. Government or anyone licensing or using the programs on behalf of the U.S. Government, the following notice is applicable:

Restricted Rights Notice Programs delivered subject to the DOD FAR Supplement are "commercial computer software" and use, duplication, and disclosure of the Programs, including documentation, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement. Otherwise, Programs delivered subject to the Federal Acquisition Regulations are "restricted computer software" and use, duplication, and disclosure of the Programs shall be subject to the restrictions in FAR 52.227-19, Commercial Computer Software - Restricted Rights (June, 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and Oracle Corporation disclaims liability for any damages caused by such use of the Programs.

Oracle is a registered trademark, and Oracle9i, Oracle Store, SQL*Plus, and PL/SQL are trademarks or registered trademarks of Oracle Corporation. Other names may be trademarks of their respective owners.

Contents

Send Us Your Comments	xvii
Preface.....	xix
Audience	xx
Organization.....	xx
Related Documentation	xxiv
Conventions.....	xxv
Documentation Accessibility	xxix
Part I Streams Concepts	
1 Introduction to Streams	
Streams Overview.....	1-2
What Can Streams Do?	1-3
Why Use Streams?.....	1-4
Capture Process Overview	1-6
Event Staging and Propagation Overview.....	1-7
Directed Networks Overview.....	1-8
Explicit Enqueue and Dequeue of Events.....	1-9
Apply Process Overview	1-11
Automatic Conflict Detection and Resolution	1-12

Rules Overview	1-13
Table Rules Overview	1-14
Schema Rules Overview	1-14
Global Rules Overview	1-14
Transformations Overview	1-15
Heterogeneous Information Sharing Overview	1-16
Oracle to Non-Oracle Data Sharing Overview.....	1-16
Non-Oracle to Oracle Data Sharing Overview.....	1-18
Example Streams Configurations	1-19
Administration Tools for a Streams Environment	1-21
Oracle-Supplied PL/SQL Packages	1-21
Streams Data Dictionary Views.....	1-22
Streams Tool in Oracle Enterprise Manager	1-23

2 Streams Capture Process

The Redo Log and the Capture Process	2-2
Logical Change Records (LCRs)	2-2
Row LCRs	2-3
DDL LCRs	2-4
Capture Rules	2-5
Datatypes Captured	2-6
Types of Changes Captured	2-7
Types of DML Changes Captured.....	2-7
Types of DDL Changes Ignored by a Capture Process.....	2-8
Other Types of Changes Ignored by a Capture Process	2-9
NOLOGGING and UNRECOVERABLE Keywords for SQL Operations.....	2-9
UNRECOVERABLE Clause for Direct Path Loads.....	2-10
Supplemental Logging in a Streams Environment	2-11
Instantiation	2-13
The Start SCN, Captured SCN, and Applied SCN for a Capture Process	2-15
Start SCN.....	2-15
Captured SCN	2-16
Applied SCN	2-16
Streams Capture Processes and RESTRICTED SESSION	2-16
Streams Capture Processes and Oracle Real Application Clusters	2-17

Capture Process Architecture	2-18
Capture Process Components.....	2-19
LogMiner Configuration	2-20
Capture Process Creation.....	2-21
ARCHIVELOG Mode and a Capture Process	2-25
Capture Process Parameters	2-25
Capture Process Rule Evaluation.....	2-27
The Persistent State of a Capture Process	2-30

3 Streams Staging and Propagation

Event Staging and Propagation Overview	3-2
Captured and User-Enqueued Events	3-3
Event Propagation Between Queues	3-4
Propagation Rules	3-5
Ensured Event Delivery.....	3-6
Directed Networks	3-7
SYS.AnyData Queues and User Messages	3-11
SYS.AnyData Wrapper for User Messages Payloads.....	3-11
Programmatic Environments for Enqueue and Dequeue of User Messages.....	3-12
Message Propagation and SYS.AnyData Queues.....	3-16
User-Defined Type Messages	3-17
Streams Queues and Oracle Real Application Clusters	3-18
Streams Staging and Propagation Architecture	3-19
Queue Buffers.....	3-19
Propagation Jobs.....	3-20
Secure Queues.....	3-22
Transactional and Nontransactional Queues	3-24
Streams Data Dictionary for Propagations	3-25

4 Streams Apply Process

Apply Process Overview	4-2
Apply Rules	4-2
Event Processing with an Apply Process	4-3
Processing Captured and User-Enqueued Events with an Apply Process.....	4-3
Event Processing Options.....	4-4

Datatypes Applied	4-9
Considerations for Applying DML Changes to Tables	4-10
Constraints	4-10
Substitute Key Columns	4-11
Row Subsetting Using Streams Rules	4-12
Apply Process Behavior for Column Discrepancies	4-14
Conflict Resolution and an Apply Process	4-16
Handlers and Row LCR Processing	4-16
Considerations for Applying DDL Changes	4-21
Types of DDL Changes Ignored by an Apply Process	4-21
Database Structures in a Streams Environment	4-22
Current Schema User Must Exist at Destination Database	4-23
System-Generated Names	4-23
CREATE TABLE AS SELECT Statements	4-24
Trigger Firing Property	4-25
Instantiation SCN and Ignore SCN	4-27
The Oldest SCN for an Apply Process	4-28
Low-Watermark and High-Watermark for an Apply Process	4-28
Streams Apply Processes and RESTRICTED SESSION	4-29
Streams Apply Processes and Oracle Real Application Clusters	4-29
Apply Process Architecture	4-30
Apply Process Components	4-31
Apply Process Creation	4-32
Streams Data Dictionary for an Apply Process	4-33
Apply Process Parameters	4-34
The Persistent State of an Apply Process	4-36
Exception Queues	4-37

5 Rules

The Components of a Rule	5-2
Rule Condition	5-2
Rule Evaluation Context	5-5
Rule Action Context	5-9

Rule Set Evaluation	5-12
Rule Set Evaluation Process	5-13
Partial Evaluation	5-14
Database Objects and Privileges Related to Rules	5-15
Privileges for Creating Database Objects Related to Rules	5-17
Privileges for Altering Database Objects Related to Rules.....	5-17
Privileges for Dropping Database Objects Related to Rules	5-18
Privileges for Placing Rules in a Rule Set.....	5-18
Privileges for Evaluating a Rule Set.....	5-19
Privileges for Using an Evaluation Context	5-19

6 How Rules Are Used In Streams

Overview of How Rules Are Used In Streams	6-2
System-Created Rules	6-3
Table and Subset Rules	6-6
Schema Rules.....	6-12
Global Rules	6-14
Streams Evaluation Context	6-15
Streams and Event Contexts	6-18
Streams and Action Contexts	6-18
User-Created Rules, Rule Sets, and Evaluation Contexts	6-19
Complex Rule Conditions	6-20
Custom Evaluation Contexts	6-24
Rule-Based Transformations	6-25
Rule-Based Transformations and a Capture Process	6-28
Rule-Based Transformations and a Propagation	6-30
Rule-Based Transformations and an Apply Process	6-32
Multiple Rule-Based Transformations	6-34

7 Streams Conflict Resolution

About DML Conflicts in a Streams Environment	7-2
Conflict Types in a Streams Environment	7-2
Conflicts and Transaction Ordering in a Streams Environment	7-4
Conflict Detection in a Streams Environment	7-5

Conflict Avoidance in a Streams Environment	7-6
Use a Primary Database Ownership Model	7-6
Avoid Specific Types of Conflicts.....	7-6
Conflict Resolution in a Streams Environment	7-8
Prebuilt Update Conflict Handlers	7-8
Custom Conflict Handlers.....	7-15

8 Streams Tags

Introduction to Tags	8-2
Tags and Rules Created by the DBMS_STREAMS_ADM Package	8-3
Tags and an Apply Process	8-6
Avoid Change Cycling with Tags	8-8
Each Databases Is a Source and Destination Database for Shared Data	8-8
Primary Database Sharing Data with Several Secondary Databases	8-12
Primary Database Sharing Data with Several Extended Secondary Databases	8-19

9 Streams Heterogeneous Information Sharing

Oracle to Non-Oracle Data Sharing with Streams	9-2
Change Capture and Staging in an Oracle to Non-Oracle Environment	9-3
Change Apply in an Oracle to Non-Oracle Environment	9-3
Transformations in an Oracle to Non-Oracle Environment	9-9
Messaging Gateway and Streams.....	9-9
Error Handling in an Oracle to Non-Oracle Environment	9-10
Example Oracle to Non-Oracle Streams Environment	9-10
Non-Oracle to Oracle Data Sharing with Streams	9-10
Change Capture and Staging in a Non-Oracle to Oracle Environment	9-11
Change Apply in a Non-Oracle to Oracle Environment.....	9-12
Instantiation from a Non-Oracle Database to an Oracle Database.....	9-12
Non-Oracle to Non-Oracle Data Sharing with Streams	9-13

10 Streams High Availability Environments

Overview of Streams High Availability Environments	10-2
Protection from Failures	10-2
Streams Replica Database	10-3
When Not to Use Streams	10-5
Application Maintained Copies	10-6
Best Practices for Streams High Availability Environments	10-7
Configuring Streams for High Availability	10-7
Recovering from Failures	10-9

Part II Streams Administration

11 Configuring a Streams Environment

Configuring a Streams Administrator	11-2
Setting Initialization Parameters Relevant to Streams	11-4
Setting Export and Import Parameters Relevant to Streams	11-8
Export Utility Parameters Relevant to Streams	11-9
Import Utility Parameters Relevant to Streams	11-10
Configuring a Database to Run a Streams Capture Process	11-13
Configuring the Database to Run in ARCHIVELOG Mode	11-13
Specifying an Alternate Tablespace for LogMiner	11-13
Configuring Network Connectivity and Database Links	11-14
Configuring a Capture-Based Streams Environment	11-15
Creating a New Streams Single Source Environment	11-15
Adding Shared Objects to an Existing Single Source Environment	11-19
Adding a New Destination Database to an Existing Single Source Environment	11-22
Creating a New Multiple Source Environment	11-25
Adding Shared Objects to an Existing Multiple Source Environment	11-30
Adding a New Database to an Existing Multiple Source Environment	11-35

12 Managing a Capture Process

Creating a Capture Process	12-2
Example of Creating a Capture Process Using DBMS_STREAMS_ADM	12-3
Example of Creating a Capture Process Using DBMS_CAPTURE_ADM	12-4

Starting a Capture Process	12-5
Specifying the Rule Set for a Capture Process	12-5
Adding Rules to the Rule Set for a Capture Process	12-5
Removing a Rule from the Rule Set for a Capture Process	12-6
Removing the Rule Set for a Capture Process	12-7
Setting a Capture Process Parameter	12-8
Specifying Supplemental Logging at a Source Database	12-9
Specifying Table Supplemental Logging Using Unconditional Log Groups	12-9
Specifying Table Supplemental Logging Using Conditional Log Groups.....	12-9
Dropping a Supplemental Log Group.....	12-10
Specifying Database Supplemental Logging of Key Columns	12-10
Dropping Database Supplemental Logging of Key Columns.....	12-10
Setting the Start SCN for a Capture Process	12-11
Preparing Database Objects for Instantiation at a Source Database	12-11
Aborting Preparation for Instantiation at a Source Database	12-12
Changing the DBID of a Database Where Changes Are Captured	12-13
Resetting the Log Sequence Number Where Changes Are Captured	12-13
Stopping a Capture Process	12-14
Dropping a Capture Process	12-14

13 Managing Staging and Propagation

Managing Streams Queues	13-2
Creating a Streams Queue	13-2
Enabling a User to Perform Operations on a Secure Queue	13-3
Disabling a User from Performing Operations on a Secure Queue	13-5
Dropping a Streams Queue	13-7
Managing Streams Propagations and Propagation Jobs	13-7
Creating a Propagation	13-8
Enabling a Propagation Job	13-11
Scheduling a Propagation Job	13-11
Altering the Schedule of a Propagation Job.....	13-12
Unschedulering a Propagation Job.....	13-13
Specifying the Rule Set for a Propagation.....	13-14
Adding Rules to the Rule Set for a Propagation	13-14
Removing a Rule from the Rule Set for a Propagation	13-16

Removing the Rule Set for a Propagation.....	13-16
Disabling a Propagation Job.....	13-17
Dropping a Propagation.....	13-18
Managing a Streams Messaging Environment	13-18
Wrapping User Message Payloads in a SYS.AnyData Wrapper	13-19
Propagating Messages Between a SYS.AnyData Queue and a Typed Queue	13-24

14 Managing an Apply Process

Creating, Starting, Stopping, and Dropping an Apply Process.....	14-2
Creating an Apply Process.....	14-2
Starting an Apply Process	14-7
Stopping an Apply Process	14-7
Dropping an Apply Process.....	14-7
Managing the Rule Set for an Apply Process.....	14-8
Specifying the Rule Set for an Apply Process	14-8
Adding Rules to the Rule Set for an Apply Process	14-8
Removing a Rule from the Rule Set for an Apply Process.....	14-9
Removing the Rule Set for an Apply Process.....	14-10
Setting an Apply Process Parameter	14-11
Setting the Apply User for an Apply Process.....	14-12
Managing the Message Handler for an Apply Process	14-13
Setting the Message Handler for an Apply Process	14-13
Removing the Message Handler for an Apply Process	14-13
Managing a DML Handler.....	14-14
Creating a DML Handler.....	14-14
Setting a DML Handler.....	14-16
Removing a DML Handler.....	14-18
Managing the DDL Handler for an Apply Process.....	14-18
Creating a DDL Handler for an Apply Process	14-19
Setting the DDL Handler for an Apply Process.....	14-20
Removing the DDL Handler for an Apply Process.....	14-21
Managing an Error Handler.....	14-21
Creating an Error Handler	14-21
Setting an Error Handler	14-26
Removing an Error Handler	14-27

Managing the Substitute Key Columns for a Table	14-27
Setting Substitute Key Columns for a Table	14-28
Removing the Substitute Key Columns for a Table.....	14-29
Managing Streams Conflict Resolution	14-29
Setting an Update Conflict Handler.....	14-30
Modifying an Existing Update Conflict Handler.....	14-31
Removing an Existing Update Conflict Handler	14-32
Managing Apply Errors	14-33
Retrying Apply Error Transactions.....	14-33
Deleting Apply Error Transactions.....	14-34
Setting Instantiation SCNs at a Destination Database	14-35
Setting Instantiation SCNs Using Export/Import	14-36
Setting Instantiation SCNs Using the DBMS_APPLY_ADM Package	14-38

15 Managing Rules and Rule-Based Transformations

Managing Rule Sets and Rules	15-2
Creating a Rule Set	15-2
Creating a Rule.....	15-3
Adding a Rule to a Rule Set	15-5
Altering a Rule	15-5
Modifying System-Created Rules	15-6
Removing a Rule from a Rule Set.....	15-7
Dropping a Rule.....	15-7
Dropping a Rule Set	15-8
Managing Privileges on Evaluation Contexts, Rule Sets, and Rules	15-8
Granting System Privileges on Evaluation Contexts, Rule Sets, and Rules.....	15-9
Granting Object Privileges on an Evaluation Context, Rule Set, or Rule	15-9
Revoking System Privileges on Evaluation Contexts, Rule Sets, and Rules.....	15-10
Revoking Object Privileges on an Evaluation Context, Rule Set, or Rule	15-10
Managing Rule-Based Transformations	15-11
Creating a Rule-Based Transformation	15-11
Altering a Rule-Based Transformation.....	15-18
Removing a Rule-Based Transformation	15-21

16 Other Streams Management Tasks

Managing Logical Change Records (LCRs)	16-2
Constructing and Enqueuing LCRs	16-2
The use_old Parameter in Some Row LCR Member Functions.....	16-8
Constructing and Processing LCRs Containing LOB Columns.....	16-11
Managing Streams Tags	16-26
Managing Streams Tags for the Current Session.....	16-26
Managing Streams Tags for an Apply Process.....	16-27
Performing Database Point-in-Time Recovery on a Destination Database	16-29
Resetting the Start SCN for the Existing Capture Process to Perform Recovery	16-30
Creating a New Capture Process to Perform Recovery	16-32
Performing Full Database Export/Import on a Database Using Streams	16-35

17 Monitoring a Streams Environment

Summary of Streams Static Data Dictionary Views	17-2
Summary of Streams Dynamic Performance Views	17-3
Monitoring a Streams Capture Process	17-3
Displaying the Queue, Rule Set, and Status of Each Capture Process	17-3
Displaying General Information About a Capture Process	17-4
Listing the Parameter Settings for a Capture Process	17-6
Determining the Applied SCN for All Capture Processes in a Database.....	17-7
Determining Redo Log Scanning Latency for a Capture Process	17-7
Determining Event Enqueuing Latency for a Capture Process	17-8
Determining Which Database Objects Are Prepared for Instantiation	17-9
Displaying Supplemental Log Groups at a Source Database	17-11
Monitoring a Streams Queue	17-12
Displaying the Streams Queues in a Database.....	17-12
Determining the Consumer of Each User-Enqueued Event in a Queue	17-13
Viewing the Contents of User-Enqueued Events in a Queue	17-13
Monitoring Streams Propagations and Propagation Jobs	17-15
Determining the Source Queue and Destination Queue for a Propagation	17-15
Determining the Rule Set for a Propagation.....	17-16
Displaying the Schedule for a Propagation Job	17-17
Determining the Total Number of Events and Bytes Propagated.....	17-19

Monitoring a Streams Apply Process	17-20
Displaying General Information About Each Apply Process	17-21
Listing the Parameter Settings for an Apply Process	17-22
Displaying Information About Apply Handlers.....	17-23
Displaying the Substitute Key Columns Specified at a Destination Database	17-25
Displaying Information About Update Conflict Handlers for a Destination Database.	17-26
Determining the Tables for Which an Instantiation SCN Has Been Set	17-27
Displaying Information About the Reader Server for an Apply Process	17-28
Determining Capture to Dequeue Latency for an Event	17-29
Displaying Information About the Coordinator Process.....	17-30
Determining the Capture to Apply Latency for an Event	17-31
Displaying Information About the Apply Servers for an Apply Process	17-33
Displaying Effective Apply Parallelism for an Apply Process	17-35
Checking for Apply Errors	17-36
Displaying Detailed Information About Apply Errors.....	17-37
Monitoring Rules and Rule-Based Transformations	17-42
Displaying the Streams Rules Used by a Streams Process or Propagation.....	17-43
Displaying the Condition for a Streams Rule	17-44
Displaying the Evaluation Context for Each Rule Set	17-45
Displaying Information About the Tables Used by an Evaluation Context	17-45
Displaying Information About the Variables Used in an Evaluation Context.....	17-46
Displaying All of the Rules in a Rule Set	17-47
Displaying the Condition for Each Rule in a Rule Set.....	17-48
Listing Each Rule that Contains a Specified Pattern in Its Condition.....	17-49
Displaying the Rule-Based Transformations in a Rule Set	17-49
Monitoring Streams Tags	17-50
Displaying the Tag Value for the Current Session.....	17-50
Displaying the Tag Value for an Apply Process	17-51

18 Troubleshooting a Streams Environment

Troubleshooting Capture Problems	18-2
Is the Capture Process Enabled?.....	18-2
Is the Capture Process Current?	18-3
Is LOG_PARALLELISM Set to 1?.....	18-3
Is LOGMNR_MAX_PERSISTENT_SESSIONS Set High Enough?	18-4

Troubleshooting Propagation Problems	18-4
Does the Propagation Use the Correct Source and Destination Queue?.....	18-5
Is the Propagation Job Used by a Propagation Enabled?	18-6
Are There Enough Job Queue Processes?	18-7
Is Security Configured Properly for the Streams Queue?	18-8
Troubleshooting Apply Problems	18-9
Is the Apply Process Enabled?.....	18-10
Is the Apply Process Current?	18-10
Does the Apply Process Apply Captured Events or User-Enqueued Events?.....	18-11
Is a Custom Apply Handler Specified?	18-12
Is the Apply Process Waiting for a Dependent Transaction?	18-12
Are There Any Apply Errors in the Exception Queue?	18-13
Troubleshooting Problems with Rules and Rule-Based Transformations	18-17
Are Rules Configured Properly for the Streams Process or Propagation?	18-17
Are the Rule-Based Transformations Configured Properly?	18-23
Checking the Trace Files and Alert Log for Problems	18-24
Does a Capture Process Trace File Contain Messages About Capture Problems?	18-25
Do the Trace Files Related to Propagation Jobs Contain Messages About Problems? ..	18-25
Does an Apply Process Trace File Contain Messages About Apply Problems?	18-26

Part III Example Environments and Applications

19 Streams Messaging Example

Overview of Messaging Example	19-2
Prerequisites	19-4
Set Up Users and Create a Streams Queue	19-5
Create the Enqueue Procedures	19-10
Configure an Apply Process	19-15
Configure Explicit Dequeue	19-23
Enqueue Events	19-28
Dequeue Events Explicitly and Query for Applied Events	19-34
Enqueue and Dequeue Events Using JMS	19-35

20 Single Database Capture and Apply Example

Overview of the Single Database Capture and Apply Example	20-2
Prerequisites.....	20-4
Set Up the Environment	20-5
Configure Capture and Apply	20-10
Make DML Changes, Query for Results, and Dequeue Events	20-22

21 Simple Single Source Replication Example

Overview of the Simple Single Source Replication Example.....	21-2
Prerequisites.....	21-3
Set Up Users and Create Queues and Database Links	21-4
Configure Capture, Propagation, and Apply for Changes to One Table.....	21-10
Make Changes to the hr.jobs Table and View Results	21-18

22 Single Source Heterogeneous Replication Example

Overview of the Single Source Heterogeneous Replication Example	22-2
Prerequisites.....	22-5
Set Up Users and Create Queues and Database Links	22-7
Example Scripts for Sharing Data from One Database	22-20
Simple Configuration for Sharing Data from a Single Database	22-21
Flexible Configuration for Sharing Data from a Single Database	22-40
Make DML and DDL Changes to Tables in the hr Schema	22-62
Add Objects to an Existing Streams Replication Environment.....	22-64
Make a DML Change to the hr.employees Table.....	22-75
Add a Database to an Existing Streams Replication Environment	22-76
Make a DML Change to the hr.departments Table	22-90

23 Multiple Source Replication Example

Overview of the Multiple Source Databases Example.....	23-2
Prerequisites.....	23-5
Set Up Users and Create Queues and Database Links	23-6
Example Script for Sharing Data from Multiple Databases.....	23-25
Make DML and DDL Changes to Tables in the hr Schema	23-62

24 Rule-Based Application Example

Overview of the Rule-Based Application	24-2
Using Rules on Non-Table Data Stored in Explicit Variables	24-3
Using Rules on Data Stored in a Table	24-9
Using Rules on Both Explicit Variables and Table Data	24-18
Using Rules on Implicit Variables and Table Data	24-27

Part IV Appendixes

A XML Schema for LCRs

Definition of the XML Schema for LCRs	A-2
--	-----

Index

Send Us Your Comments

Oracle9i Streams, Release 2 (9.2)

Part No. A96571-02

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

Oracle9i Streams describes the features and functionality of Streams. This document contains conceptual information about Streams, along with information about configuring and managing a Streams environment. In addition, this document contains detailed examples for configuring a Streams messaging environment, a Streams replication environment, and a rule-based application.

This preface contains these topics:

- [Audience](#)
- [Organization](#)
- [Related Documentation](#)
- [Conventions](#)
- [Documentation Accessibility](#)

Audience

Oracle9i Streams 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
- Configure conflict resolution in 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-level rules, subset rules, schema-level rules, and global-level rules. Also includes information about rule-based transformations.

Chapter 7, "Streams Conflict Resolution"

Contains conceptual information about conflicts. Includes information about the possible types of conflicts, conflict detection, conflict avoidance, and conflict resolution in Streams environments.

Chapter 8, "Streams Tags"

Contains conceptual information about Streams tags. Includes information about how tag values are used in rules, how a tag value can be set for an apply process, and how to avoid change cycling using tags.

Chapter 9, "Streams Heterogeneous Information Sharing"

Contains conceptual information about heterogeneous information sharing using Streams. Includes information about sharing information in an Oracle database with a non-Oracle database, sharing information in a non-Oracle database with an Oracle database, and using Streams to share information between two non-Oracle databases.

Chapter 10, "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 11, "Configuring 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 12, "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 13, "Managing Staging and Propagation"

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

Chapter 14, "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 15, "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 16, "Other Streams Management Tasks"

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

Chapter 17, "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 18, "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, and Streams rules, 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 19, "Streams Messaging Example"

Contains a step by step example that configures a messaging environment using Streams.

Chapter 20, "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 21, "Simple Single Source Replication Example"

Contains a step by step example that configures a simple single source replication environment using Streams.

Chapter 22, "Single Source Heterogeneous Replication Example"

Contains a step by step example that configures a single source heterogeneous replication environment using Streams. Also contains step by step examples for adding objects and databases to this environment.

Chapter 23, "Multiple Source Replication Example"

Contains a step by step example that configures a multiple source replication environment using Streams.

Chapter 24, "Rule-Based Application Example"

Contains step by step examples that illustrate a rule-based application that uses the Oracle rules engine.

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.

Related Documentation

For more information, see these Oracle resources:

- *Oracle9i Database Concepts*
- *Oracle9i Database Administrator's Guide*
- *Oracle9i SQL Reference*
- *Oracle9i Supplied PL/SQL Packages and Types Reference*
- *PL/SQL User's Guide and Reference*
- *Oracle9i Database Utilities*
- *Oracle9i Heterogeneous Connectivity Administrator's Guide*
- *Oracle9i Application Developer's Guide - Advanced Queuing*
- 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 Oracle9i documentation set.

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 *Oracle9i Sample Schemas* for information on how these schemas were created and how you can use them.

In North America, printed documentation is available for sale in the Oracle Store at <http://oraclestore.oracle.com/>

Other customers can contact their Oracle representative to purchase printed documentation.

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN) at

<http://otn.oracle.com/>

You must register online before using OTN. 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/docs/index.htm>

To access the database documentation search engine directly, please visit

<http://tahiti.oracle.com>

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 .
<i>Italics</i>	Italic typeface indicates book titles or emphasis.	<i>Oracle9i Database Concepts</i> Ensure that the recovery catalog and target database do <i>not</i> 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.
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.
<i>lowercase italic monospace (fixed-width) font</i>	Lowercase italic monospace font represents placeholders or variables.	You can specify the <i>parallel_clause</i> . Run <i>Uold_release</i> .SQL where <i>old_release</i> 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 (<i>digits</i> [, <i>precision</i>])
{ }	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]
...	Horizontal ellipsis points indicate either: <ul style="list-style-type: none"> 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 <i>subquery</i> ; SELECT <i>col1</i> , <i>col2</i> , ... , <i>coln</i> 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 /fsl/dbs/tbs_02.dbf
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;

Convention	Meaning	Example
<i>Italics</i>	Italicized text indicates placeholders or variables for which you must supply particular values.	CONNECT SYSTEM/ <i>system_password</i> DB_NAME = <i>database_name</i>
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 Corporation 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 Corporation does not own or control. Oracle Corporation neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Part I

Streams Concepts

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

- [Chapter 1, "Introduction to Streams"](#)
- [Chapter 2, "Streams Capture Process"](#)
- [Chapter 3, "Streams Staging and Propagation"](#)
- [Chapter 4, "Streams Apply Process"](#)
- [Chapter 5, "Rules"](#)
- [Chapter 6, "How Rules Are Used In Streams"](#)
- [Chapter 7, "Streams Conflict Resolution"](#)
- [Chapter 8, "Streams Tags"](#)
- [Chapter 9, "Streams Heterogeneous Information Sharing"](#)
- [Chapter 10, "Streams High Availability Environments"](#)

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.

This chapter contains these topics:

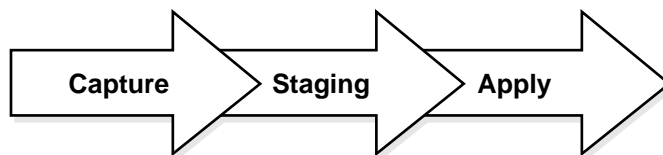
- [Streams Overview](#)
- [Capture Process Overview](#)
- [Event Staging and Propagation Overview](#)
- [Apply Process Overview](#)
- [Automatic Conflict Detection and Resolution](#)
- [Rules Overview](#)
- [Transformations Overview](#)
- [Heterogeneous Information Sharing Overview](#)
- [Example Streams Configurations](#)
- [Administration Tools for a Streams Environment](#)

Streams Overview

Oracle Streams enables you to share data and events in a stream. The stream can propagate this information within a database or from one database to another. The stream routes specified information to specified destinations. The result is a new 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 enables you to break the cycle of trading off one solution for another. Streams provides the capabilities needed to build and operate distributed enterprises and applications, data warehouses, and high availability solutions. You can use all 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 can also put user-defined events into a stream. Then, Streams can propagate the information to other databases or applications automatically. Again, based on your specifications, Streams can apply events at a destination database. [Figure 1-1](#) shows the Streams information flow.

Figure 1-1 *Streams Information Flow*



What Can Streams Do?

You can use Streams to:

- **Capture** changes at a database.

You can configure a background **capture process** to capture changes made to tables, schemas, or the entire database. A capture process captures changes from the redo log and formats each captured change into a logical change record (**LCR**). The database where changes are generated in the redo log is called the **source database**.

- Enqueue **events** into a queue. Two types of events may be staged in a Streams queue: LCRs and user messages.

A capture process enqueues LCR events into a queue that you specify. The queue can then share the LCR events within the same database or with other databases.

You can also enqueue user events explicitly with a user application. These explicitly enqueued events can be LCRs or user messages.

- Propagate events from one queue to another. These queues may be in the same database or in different databases.
- Dequeue events.

A background **apply process** can dequeue events. You can also dequeue events explicitly with a user application.

- **Apply** events at a database.

You can configure an apply process to apply all of the events in a queue or only the events that you specify. You can also configure an apply process to call your own PL/SQL subprograms to process events.

The database where LCR events are applied and other types of events are processed is called the **destination database**. In some configurations, the source database and the destination database may be the same.

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.

Why Use Streams?

The following sections briefly describe some of the reasons for using Streams.

Message Queuing

Streams allows user applications to enqueue messages of different types, propagate the messages to subscribing queues, notify user applications that messages are ready for consumption, and dequeue messages at the destination database. Streams introduces a new type of queue that stages messages of `SYS.AnyData` type. Messages of almost any type can be wrapped in a `SYS.AnyData` wrapper and staged in `SYS.AnyData` queues. Streams interoperates with Advanced Queuing (AQ), which supports all the standard features of message queuing systems, including multiconsumer queues, publishing and subscribing, content-based routing, internet propagation, transformations, and gateways to other messaging subsystems.

See Also: *Oracle9i Application Developer's Guide - Advanced Queuing* for more information about AQ

Data Replication

Streams can efficiently 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 need not 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.

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 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 allows the necessary flexibility to reformat data or update warehouse-specific data fields as data is loaded.

See Also: *Oracle9i 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 built on 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 latency sensitive queries from a production database.

The most notable difference between a logical standby database and a Streams data replication environment is where the changes are captured. 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, the exposure to lost transactions is 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.

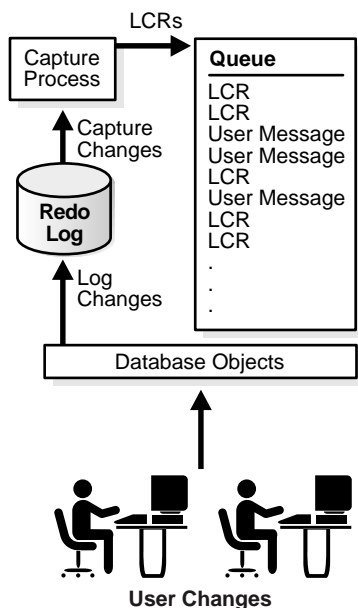
See Also:

- [Chapter 10, "Streams High Availability Environments"](#)
- *Oracle9i Data Guard Concepts and Administration* for more information about logical standby databases

Capture Process Overview

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 reads 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. You use rules to specify which changes are captured. [Figure 1-2](#) shows a capture process capturing LCRs.

Figure 1-2 The 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` or `SYSTEM` schemas.

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. A tag can be used to determine whether a redo entry or an LCR contains a change that originated in the local database or at a different database, so that you can avoid sending LCRs back to the database where they originated. Tags may be used for other LCR tracking purposes as well. You can also use tags to specify the set of destination databases for each LCR.

See Also:

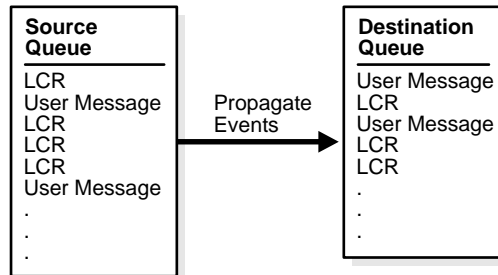
- [Chapter 2, "Streams Capture Process"](#) for more information about capture processes and for detailed information about which DML and DDL statements are captured by a capture process
- [Chapter 8, "Streams Tags"](#)

Event Staging and Propagation Overview

Streams uses queues to stage events for propagation or consumption. You can use Streams to propagate 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 the Streams apply process or by a user-defined subprogram. If you configure a propagation to propagate changes from a source queue to a destination queue, then you can use rules to specify which changes are propagated. [Figure 1-3](#) shows propagation from a source queue to a destination queue.

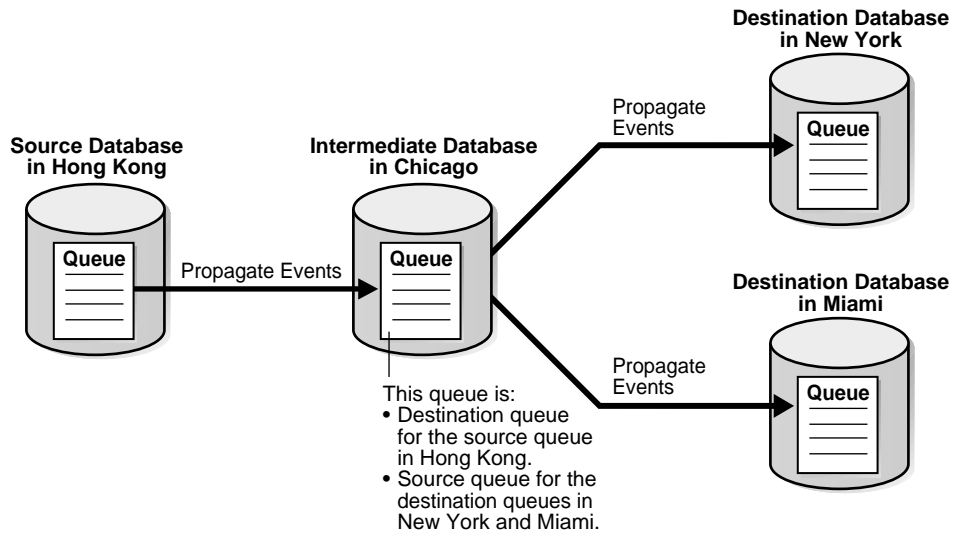
Figure 1–3 Propagation from a Source Queue to a Destination Queue



Directed Networks Overview

Streams enables you to configure an environment where changes are shared through **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. The events may or may not be processed at an intermediate 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.

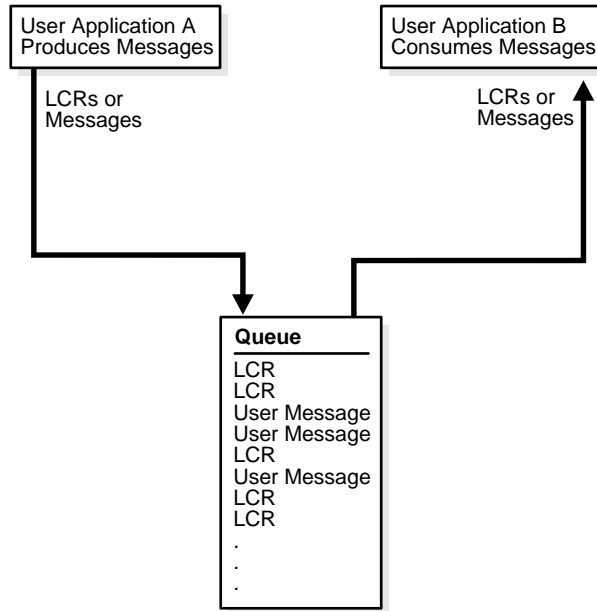
Figure 1–4 Example Directed Networks Environment

See Also: [Chapter 3, "Streams Staging and Propagation"](#) for more information about staging and propagation

Explicit Enqueue and Dequeue of Events

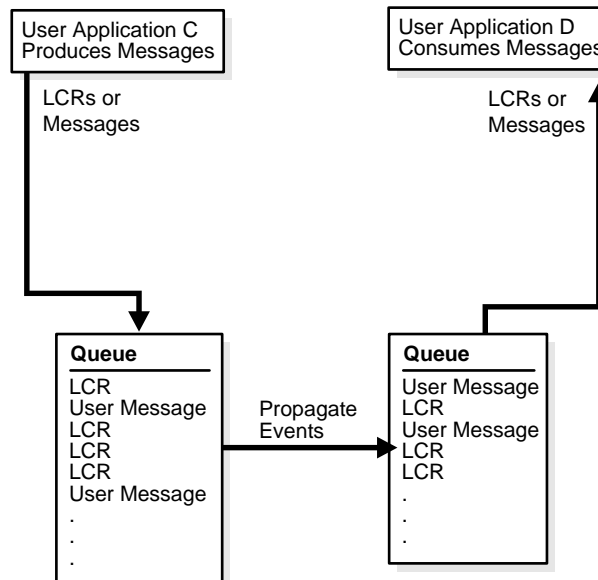
User applications can enqueue events into a queue explicitly. User applications can format these events as LCRs, which allows an apply process to apply them at a destination database. Alternatively, these events can be formatted as user messages for consumption by another user application, which either explicitly dequeues the events or processes the events with callbacks from an apply process. Events that were explicitly enqueued into a queue can be 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 explicitly enqueued into a source queue can be explicitly dequeued from a destination queue by a user application without any intervention from 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

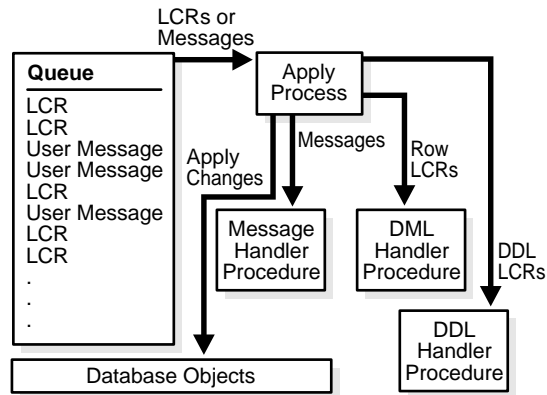


See Also: ["SYS.AnyData Queues and User Messages"](#) on page 3-11 for more information about explicit enqueue and dequeue of events

Apply Process Overview

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. These apply handlers can include message handlers, DML handlers, and DDL 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. You use rules to specify which events in the queue are applied. [Figure 1–7](#) shows an apply process processing LCRs and user messages.

Figure 1–7 The Apply Process

See Also: [Chapter 4, "Streams Apply Process"](#)

Automatic Conflict Detection and Resolution

An apply process detects conflicts automatically when directly applying LCRs. Typically, a conflict results when the same row in the source database and destination database is changed at approximately the same time.

When conflicts occur, you need a mechanism to ensure that the conflict is resolved in accordance with your business rules. Streams offers a variety of prebuilt conflict resolution 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 an exception queue for later analysis and possible reexecution.

See Also: [Chapter 7, "Streams Conflict Resolution"](#)

Rules Overview

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, and you can group related rules together into **rule sets**. A rule consists of the following components:

- The **rule condition** combines one or more expressions and operators 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 can either 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.

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

```
:dml.get_object_owner() = 'hr' AND :dml.get_object_name() = 'departments'
```

In a Streams environment, this rule condition may be used in the following ways:

- To instruct a capture process to capture DML changes to the `hr.departments` table
- To instruct a propagation to propagate DML changes to the `hr.departments` table
- To instruct an apply process to apply DML changes to the `hr.departments` table

Streams performs tasks based on rules. These tasks include capturing changes with a capture process, propagating changes with a propagation, and applying changes with an apply process. You can define rules for these tasks at three different levels:

- Table rules
- Schema rules
- Global rules

Table Rules Overview

When you define a table rule, the Streams task is performed when a change is made to the table you specify. For example, you can define a rule that instructs a capture process to capture changes to the `hr.employees` table. Given this rule, if a row is inserted into the `hr.employees` table, then the capture process captures the insert, formats it into an LCR, and enqueues the LCR into a queue.

Schema Rules Overview

When you define a schema rule, the Streams task is performed when a change is made to the database objects in the schema you specify, and any database objects added to the schema in the future. For example, you can define two rules that instruct a propagation to propagate DML and DDL changes to the `hr` schema from a source queue to a destination queue. Given these rules, suppose the source queue contains LCRs that define the following changes:

- The `hr.loc_city_ix` index is altered.
- A row is updated in the `hr.jobs` table.

The propagation propagates these changes from the source queue to the destination queue, because both changes are to database objects in the `hr` schema.

Global Rules Overview

When you define a global rule, the Streams task is performed when a change is made to any database object in the database. If it is a global DML capture rule, then a capture process captures all DML changes to the database objects in the database. If it is a global DDL propagation or apply rule, then the Streams task is performed for all DDL changes in a queue.

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` and `SYSTEM` schemas.

See Also:

- [Chapter 5, "Rules"](#)
- [Chapter 6, "How Rules Are Used In Streams"](#)

Transformations Overview

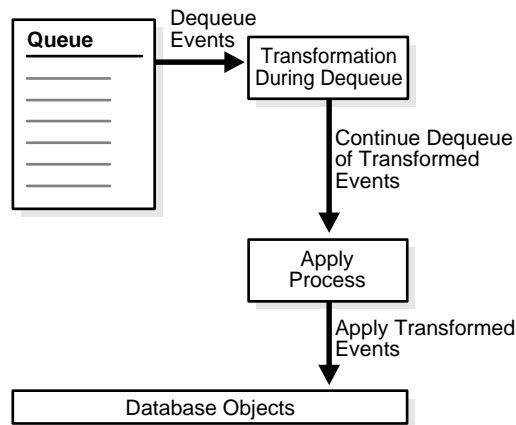
A **rule-based transformation** is any modification to an event that results when a rule evaluates to `TRUE`. For example, you can use a rule-based transformation when you want to change the datatype of a particular column in a table for an event. In this case, the transformation can be a PL/SQL function that takes as input a `SYS.AnyData` object containing a logical change record (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, 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, which can be useful for formatting an event in a manner appropriate for a specific destination database

Figure 1-8 shows a rule-based transformation during apply.

Figure 1-8 Transformation During Apply



See Also: ["Rule-Based Transformations"](#) on page 6-25

Heterogeneous Information Sharing Overview

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.

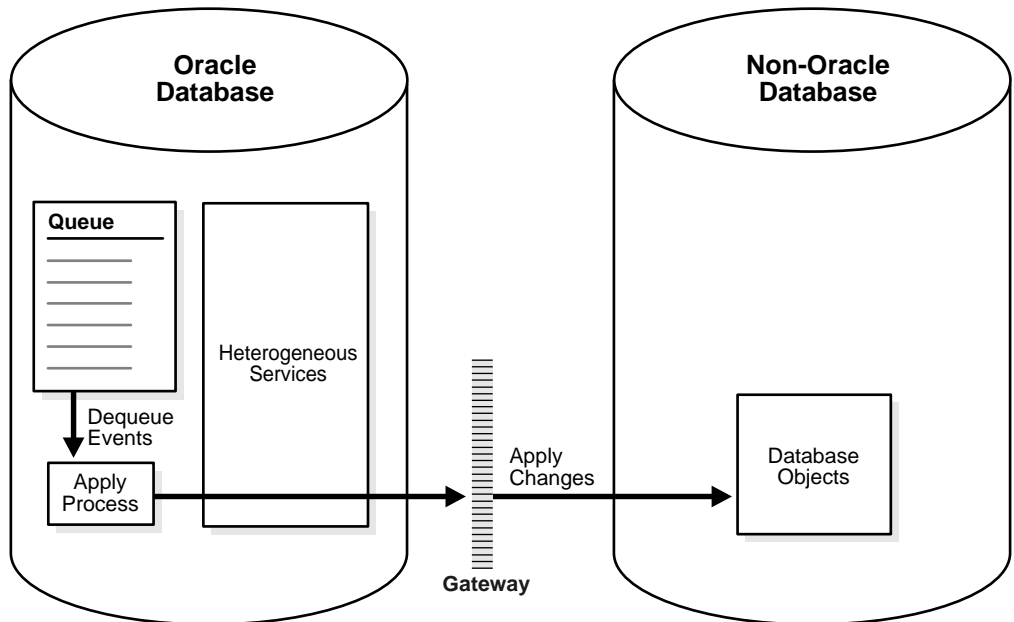
See Also: [Chapter 9, "Streams Heterogeneous Information Sharing"](#)

Oracle to Non-Oracle Data Sharing Overview

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 1-9](#) shows an Oracle databases sharing data with a non-Oracle database.

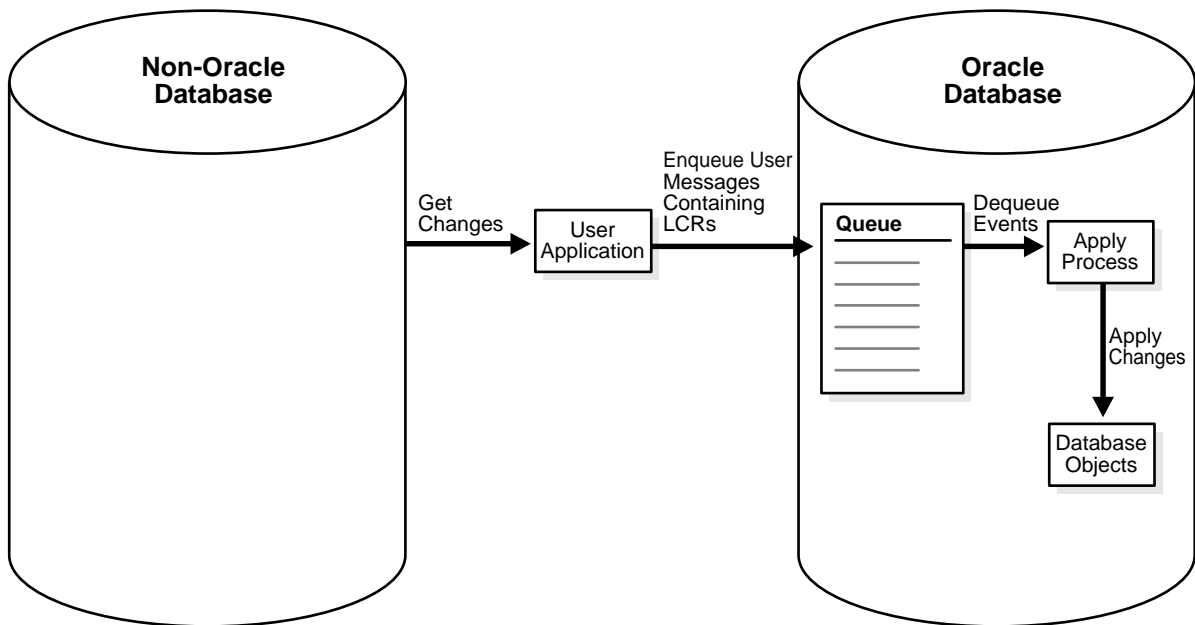
Figure 1–9 Oracle to Non-Oracle Heterogeneous Data Sharing

See Also: *Oracle9i Heterogeneous Connectivity Administrator's Guide* for more information about Heterogeneous Services

Non-Oracle to Oracle Data Sharing Overview

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 a logical change record (LCR). Then, the application must enqueue the LCRs into a queue in an Oracle database by using the PL/SQL interface, where they can be processed by an apply process. [Figure 1-10](#) shows a non-Oracle databases sharing data with an Oracle database.

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



Example Streams Configurations

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

Figure 1-11 Streams Configuration in a Single Database

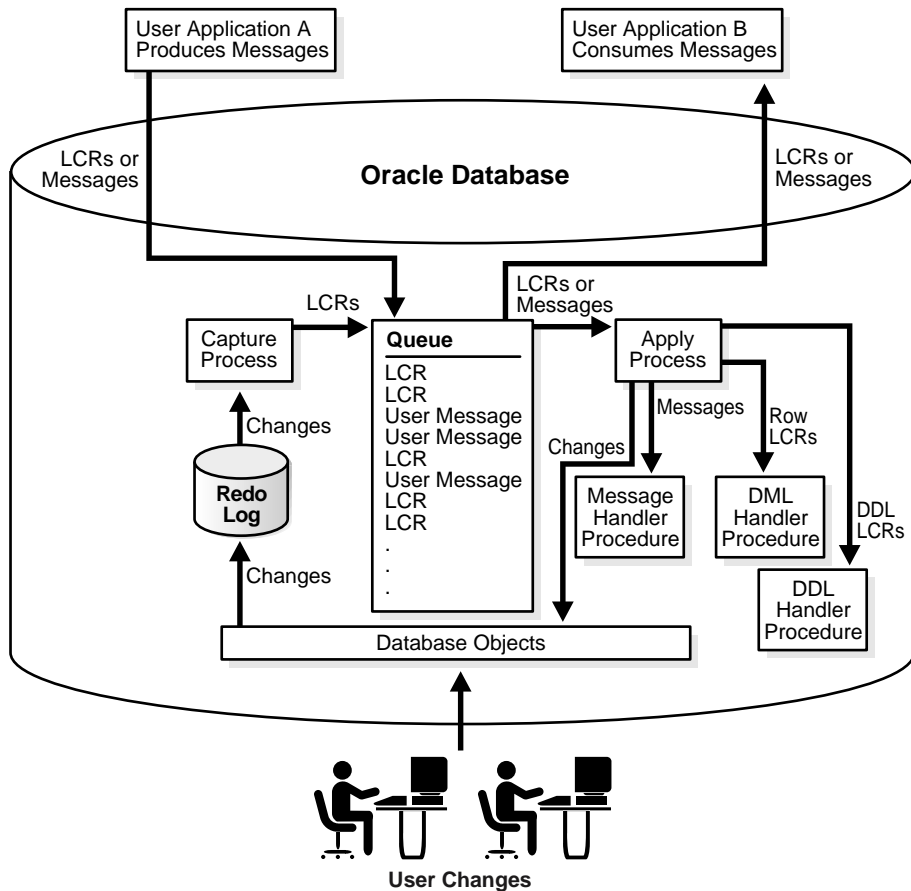
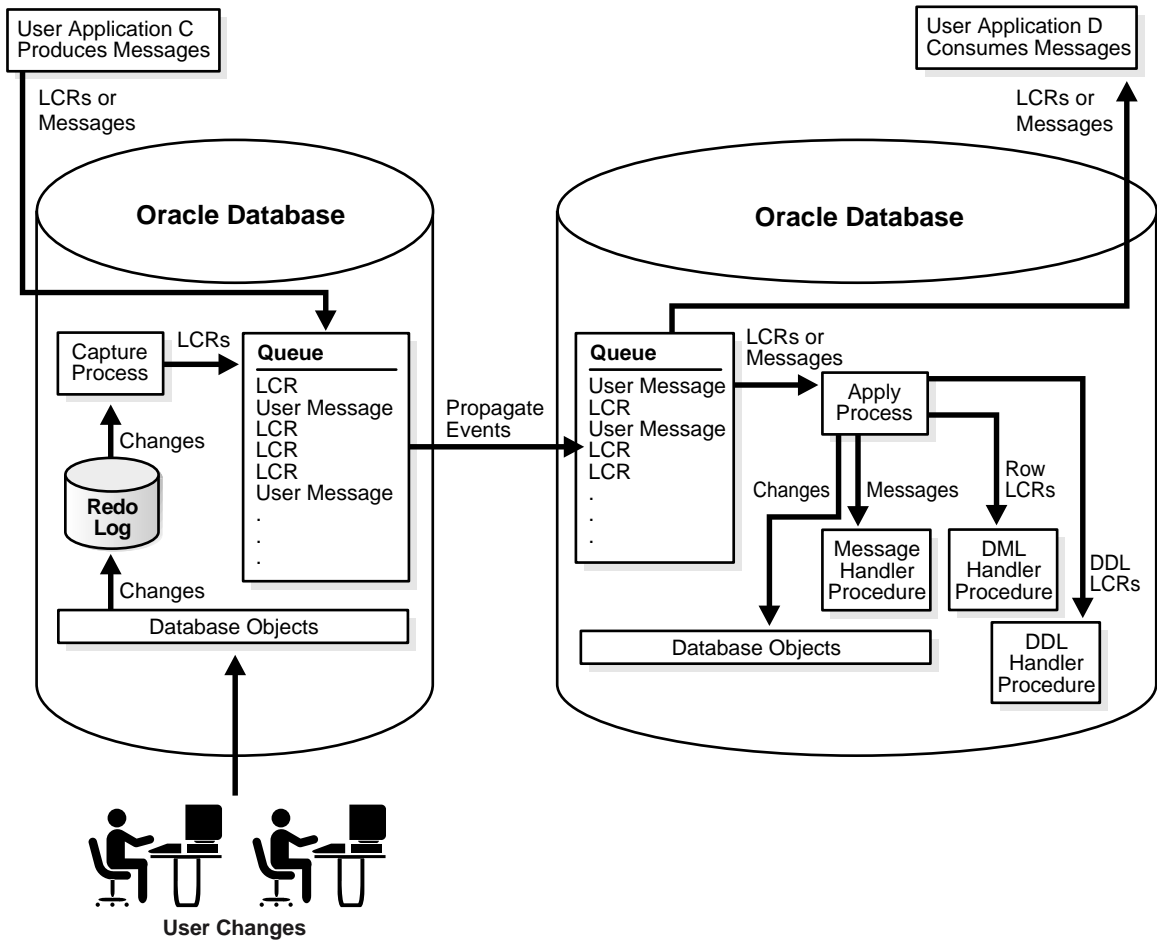


Figure 1-12 Streams Configuration Sharing Information Between Databases



Administration Tools for a Streams Environment

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

Oracle-Supplied PL/SQL Packages

The following Oracle-supplied PL/SQL packages contain procedures and functions that you can use to configure and manage a Streams environment.

See Also: *Oracle9i Supplied 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, propagation, and apply at the table, schema, and database level. This package also contains procedures for creating queues and for managing Streams metadata, such as data dictionary information. This package is provided as an easy way to complete common tasks in a Streams replication environment. You can use other packages, such as the `DBMS_CAPTURE_ADM`, `DBMS_PROPAGATION_ADM`, and `DBMS_APPLY_ADM` 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. The source of the captured changes is the redo logs, and the repository for the captured changes is a queue. 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.

DBMS_RULE_ADM Package

The `DBMS_RULE_ADM` package provides an administrative interface for creating and managing rules, rule sets, and rule evaluation contexts.

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.

DBMS_STREAMS Package

The `DBMS_STREAMS` package provides interfaces to convert `SYS.AnyData` objects into logical change record (LCR) objects, to return information about Streams attributes, and to annotate redo entries generated by a session with a binary tag. This tag affects the behavior of a capture process, a propagation job, or an apply process whose rules include specifications for these binary tags in redo entries or LCRs.

Streams Data Dictionary Views

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

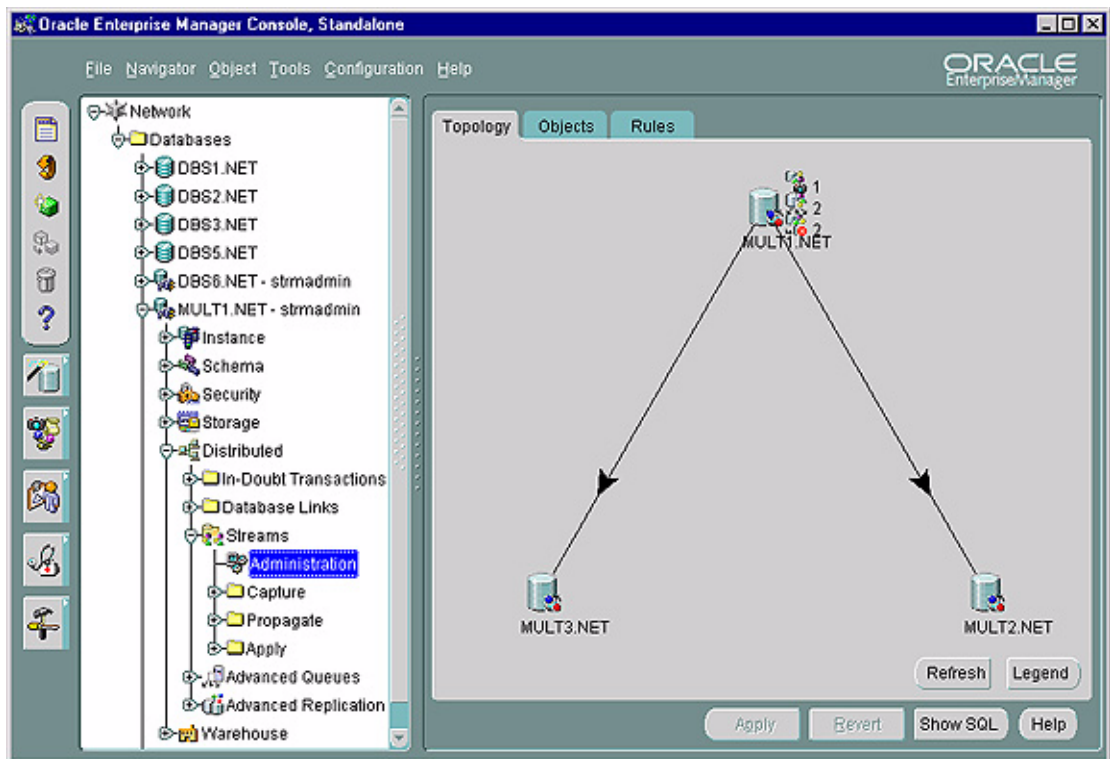
See Also:

- [Chapter 17, "Monitoring a Streams Environment"](#)
- *Oracle9i Database Reference* for more information about these data dictionary views

Streams Tool in Oracle Enterprise Manager

To help configure, administer, and monitor Streams environments, Oracle provides a Streams tool in the Oracle Enterprise Manager Console. You can also use the Streams tool to generate Streams configuration scripts, which you can then modify and run to configure your Streams environment. The Streams tool online help is the primary documentation source for this tool. [Figure 1–13](#) shows the **Topology** tab in the Streams tool.

Figure 1–13 Streams Tool



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

Streams Capture Process

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

This chapter contains these topics:

- [The Redo Log and the Capture Process](#)
- [Logical Change Records \(LCRs\)](#)
- [Capture Rules](#)
- [Datatypes Captured](#)
- [Types of Changes Captured](#)
- [Supplemental Logging in a Streams Environment](#)
- [Instantiation](#)
- [The Start SCN, Captured SCN, and Applied SCN for a Capture Process](#)
- [Streams Capture Processes and RESTRICTED SESSION](#)
- [Streams Capture Processes and Oracle Real Application Clusters](#)
- [Capture Process Architecture](#)

See Also: [Chapter 12, "Managing a Capture Process"](#)

The Redo Log and the 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.

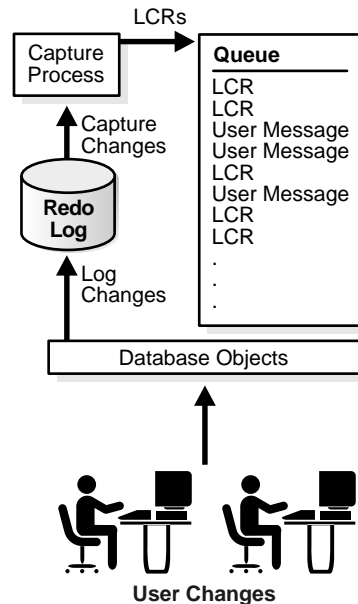
Redo logs are used to guarantee recoverability in the event of human error or media failure. A capture process is an optional Oracle background process that reads the database redo log to capture DML and DDL changes made to database objects. When a capture process is configured to capture changes from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

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.

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. 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:**

- ["Managing Logical Change Records \(LCRs\)"](#) on page 16-2
- *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about LCRs

Row LCRs

A row LCR describes a change to the data in a single row or a change to a single 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 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. So, 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. Further, the DML statement itself may be part of a transaction that includes many DML statements.

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

Each row LCR contains the following information:

- The name of the source database where the row change occurred
- The type of DML statement that produced the change, either `INSERT`, `UPDATE`, `DELETE`, `LOB ERASE`, `LOB WRITE`, or `LOB TRIM`
- The schema name that contains the table with the changed row
- The name of the table that contains the changed row
- A raw tag that can be used to track the LCR
- The identifier of the transaction in which the DML statement was run
- The system change number (SCN) when the change was written to the redo log
- The old values related to the 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.
- The new values related to the change. If the type of the DML statement is `UPDATE` or `INSERT` statement, 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.

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:

- The name of the source database where the DDL change occurred
- The type of DDL statement that produced the change, for example `ALTER TABLE` or `CREATE INDEX`
- The schema name of the user who owns the database object on which the DDL statement was run
- The name of the database object on which the DDL statement was run

-
- The type of database object on which the DDL statement was run, for example `TABLE` or `PACKAGE`
 - The text of the DDL statement
 - The logon user, which is the user whose session executed the DDL statement
 - The schema that is used if no schema is specified for an object in the DDL text
 - 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.
 - 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.
 - A raw tag that can be used to track the LCR
 - The identifier of the transaction in which the DDL statement was run
 - 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 types of DDL statements

Capture Rules

A capture process captures changes based on rules that you define. Each rule specifies the database objects for which the capture process captures changes and the types of changes to capture. You can specify capture rules at the following levels:

- A table rule captures either DML or DDL changes to a particular table.
- A schema rule captures either DML or DDL changes to the database objects in a particular schema.
- A global rule captures either all DML or all DDL changes in the database.

See Also:

- [Chapter 5, "Rules"](#)
- [Chapter 6, "How Rules Are Used In Streams"](#)

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 and SYSTEM schemas.

Datatypes Captured

When capturing changes made to tables, a capture process can capture changes made to columns of the following datatypes:

- CHAR
- VARCHAR2
- NCHAR
- NVARCHAR2
- NUMBER
- DATE
- CLOB
- BLOB
- RAW
- TIMESTAMP
- TIMESTAMP WITH TIME ZONE
- TIMESTAMP WITH LOCAL TIME ZONE
- INTERVAL YEAR TO MONTH
- INTERVAL DAY TO SECOND

The capture process raises an error when it finds a change that satisfies one of its rules to a table containing a column with a datatype that is not listed. The capture process does not capture DML changes in 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). When the 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.

See Also:

- ["Datatypes Applied"](#) on page 4-9
- *Oracle9i 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. A capture process ignores changes that it cannot capture.

Note: A capture process never captures changes in the SYS and SYSTEM schemas.

See Also: [Chapter 4, "Streams Apply Process"](#) for information about the types of changes an apply process applies and ignores

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

Note:

- The capture process converts each `MERGE` change into an `INSERT` or `UPDATE` change. `MERGE` is not a valid command type in a row LCR.
 - The capture process does not capture `CALL`, `EXPLAIN PLAN`, or `LOCK TABLE` statements.
 - The capture process cannot capture DML changes made to temporary tables, index-organized 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:

- ["Considerations for Applying DML Changes to Tables"](#) on page 4-10
- ["Avoid Uniqueness Conflicts in a Streams Environment"](#) on page 7-6 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 the rules in the capture process rule set, *except for* the following types of DDL changes:

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

A capture process captures DDL statements that satisfy the rules in the capture process rule set, but not the results of these 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: ["Considerations for Applying DDL Changes"](#) on page 4-21

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

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, if the changes resulting from a SQL operation should be captured by a capture process, then do not use these keywords.

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.

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: *Oracle9i 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.

See Also: *Oracle9i 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 UPDATE operation is performed. Such updates include piecewise updates to LOBs. The capture process captures this additional information and places it in LCRs.

There are two types of supplemental logging: database supplemental logging and table supplemental logging. Database supplemental logging specifies supplemental logging for an entire database, while table supplemental logging enables you to specify log groups for supplemental logging for a particular table. If you use table supplemental logging, then you can choose between unconditional and conditional log groups.

Unconditional log groups log the before images of specified columns any time the table is updated, regardless of whether the update affected any of the specified columns. This is sometimes referred to as an ALWAYS log group. Conditional log groups log the before images of all specified columns only if at least one of the columns in the log group is updated.

Supplementing logging at the database level, unconditional log groups at the table level, and conditional log groups at the table level together determine which old values are logged in an update statement or piecewise LOB update.

If you plan to use one or more apply processes to apply LCRs captured by a capture process, then you must enable supplemental logging *at the source database* for the following types of columns in tables *at the destination database*:

- Any columns at the source database that are used in a primary key in tables for which changes are applied at a destination database must be unconditionally logged in a log group or by database supplemental logging of primary key columns.
- If the parallelism of any apply process that will apply the changes is greater than 1, then any unique constraint at a destination database that comes from multiple columns at the source database must be conditionally logged. Supplemental logging need not be specified if the unique constraint comes from a single column at the source database.

- If the parallelism of any apply process that will apply the changes is greater than 1, then any foreign key constraint at a destination database that comes from multiple columns at the source database must be conditionally logged. Supplemental logging need not be specified if the foreign key comes from a single column at the source database.
- Any columns at the source database that are used in substitute key columns for an apply process at a destination database must be unconditionally logged. You specify substitute key columns for a table using the `SET_KEY_COLUMNS` procedure in the `DBMS_APPLY_ADM` package.
- The columns specified in a column list for conflict resolution during apply must be conditionally logged if more than one column at the source database is used in the column list at the destination database.
- Any columns at the source database that are used by a DML handler or error handler specified for update operations or piecewise updates to LOBs at a destination database must be unconditionally logged.
- Any columns at the source database that are used by a rule or a rule-based transformation must be unconditionally logged.
- If you specify row subsetting for a table at a destination database, then any columns at the source database that are in the destination table or columns at the source database that are in the subset condition must be unconditionally logged. You specify a row subsetting condition for an apply process using the `dml_condition` parameter in the `ADD_SUBSET_RULES` procedure in the `DBMS_STREAMS_ADM` package.

If you do not use supplemental logging for these types of columns at a source database, then changes involving these columns might not apply properly at a destination database.

Note: LOBs, LONGs, and user-defined types cannot be part of a supplemental log group

See Also:

- ["Specifying Supplemental Logging at a Source Database"](#) on page 12-9
- *Oracle9i Data Guard Concepts and Administration* and *Oracle9i Database Administrator's Guide* for information about using supplemental logging
- ["Considerations for Applying DML Changes to Tables"](#) on page 4-10 for more information about apply process behavior for tables
- ["Constraints"](#) on page 4-10 for more information about supplemental logging and key columns
- ["Rule-Based Transformations"](#) on page 6-25
- ["Column Lists"](#) on page 7-12 for more information about supplemental logging and column lists

Instantiation

If you plan to use a capture process to replicate changes made to a database object from a source database to a destination database, then the destination database must have a copy of the database object. If the copy does not exist at the destination database, then you must **instantiate** the object at the destination database before you can replicate changes. To instantiate an object means to create an object physically at a destination database based on an object at a source database. If the object is a table, then the objects at the source and destination database need not be an exact match, but, if some or all of the table data is replicated between the two databases, then the data that is replicated should be consistent when the table is instantiated. Typically, instantiation is done using export/import.

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. 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 may need to instantiate source database objects at destination databases. In any event, you must always prepare the object for instantiation. By preparing an object for instantiation, you are setting the earliest SCN for which changes to the object may need to be applied at the destination database.

The following procedures in the `DBMS_CAPTURE_ADM` package prepare database objects for instantiation:

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

These procedures record the lowest SCN of each object for instantiation. SCNs subsequent to the lowest SCN for an object can be used for instantiating the object. These procedures also populate the Streams data dictionary for the relevant capture processes, propagations, and apply processes that capture, propagate, or apply changes to the table, schema, or database being prepared for instantiation.

Whenever you add or modify the condition of a capture or propagation rule for an object, you must run the appropriate procedure to prepare the object for instantiation at the source database if any of the following conditions are met:

- One or more rules are added to the rule set for a capture process that instruct the capture process to capture changes made to the object. When you use the `DBMS_STREAMS_ADM` package to add rules to a rule set for a capture process, the appropriate procedure to prepare for instantiation is run automatically at the source database. When you use the `DBMS_RULE_ADM` package to add these rules, you must prepare for instantiation manually.
- One or more conditions of rules in the rule set for a capture process are modified to instruct the capture process to capture changes made to the object.
- One or more rules are added to the rule set for a propagation that instruct the propagation to propagate changes made to the object.
- One or more conditions of rules in the rule set for a propagation are modified to instruct the propagation to propagate changes made to the object.

When any of these conditions are met, you must prepare these database objects for instantiation at the source database to populate any relevant Streams data dictionary that requires information about the source object, even if the object already exists at a remote database where the rules were added or changed.

The relevant Streams data dictionaries are populated asynchronously for both the local dictionary and all remote dictionaries. The procedure that prepares for instantiation adds information to the redo log at the source database. The local Streams data dictionary is populated with the information about the object when a

capture process captures these redo entries, and any remote Streams data dictionaries are populated when the information is propagated to them.

When you instantiate tables using export/import, any table supplemental log group specifications are retained for the instantiated tables. That is, after instantiation, log group specifications for imported tables at the import database are the same as the log group specifications for these tables at the export database. If you do not want to retain supplemental log group specifications for tables at the import database, then you can drop specific supplemental log groups after import. Database supplemental logging specifications are not retained during export/import, even if you perform a full database export/import.

See Also:

- ["Preparing Database Objects for Instantiation at a Source Database"](#) on page 12-11
- ["Supplemental Logging in a Streams Environment"](#) on page 2-11 for information about table and database supplemental logging
- ["Specifying Supplemental Logging at a Source Database"](#) on page 12-9 for information about adding and dropping supplemental log groups

The Start SCN, Captured SCN, and Applied SCN for a Capture Process

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

Start SCN

The **start SCN** is the SCN from which a capture process begins to capture changes. When you start a capture process for the first time, by default the start SCN corresponds to the SCN when the capture process was created. For example, if a capture process is started two days after it was created, then the capture process begins capturing changes from the redo log at the time of creation two days in the past.

You can specify a different start SCN during capture process creation, or you can alter a capture process to set its start SCN. The start SCN value specified must be from a time after the first capture process was created for the database.

See Also:

- ["Setting the Start SCN for a Capture Process"](#) on page 12-11
- The `DBMS_CAPTURE_ADM.ALTER_CAPTURE` procedure in the *Oracle9i Supplied PL/SQL Packages and Types Reference* for information about altering a capture process

Captured SCN

The **captured SCN** is the SCN that corresponds to the most recent change captured by a capture process.

Applied SCN

The **applied SCN** for a capture process is the SCN of the most recent event dequeued by the relevant apply processes. All events below this SCN have been dequeued by all apply processes that apply changes captured by this capture process.

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 environment. If you use one or more capture processes and Real Application Clusters in the same environment, then the environment must meet the following requirements:

- All archived logs that contain changes to be captured by a capture process must be available to all instances in the Real Application Clusters environment. In a Real Application Clusters environment, a capture process always reads archived redo logs, and it reads changes made by all instances.
- Any call to the `DBMS_CAPTURE_ADM.START_CAPTURE` procedure must be run on the instance that owns the queue that is used by the capture process. Calls to other procedures and functions that operate on a capture process can be performed from any instance.
- Any supplemental logging specifications must be made on each running instance. After it is specified for each running instance, it does not need to be specified again if an instance is shut down and restarted, and it does not need to be specified for any new instances.
- The `ARCHIVE_LAG_TARGET` initialization parameter should be set to a value greater than zero. This initialization parameter specifies the duration after which the log files are switched automatically. LogMiner orders all LCRs by SCN. To do so, it needs the archived log files from all instances. Setting this parameter to switch the log files automatically ensures that LogMiner does not wait for an inordinately long time if one instance has far fewer transactions than another.

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. If this happens, then, to restart the capture process, connect to the owner instance for the queue and run the `START_CAPTURE` procedure. The `DBA_QUEUE_TABLES` data dictionary view contains information about the owner instance for a queue table. The capture process maintains a persistent start/stop state in a Real Application Clusters environment only if the owner instance for its queue does not change before the database instance owning the queue is restarted.

Also, any parallel execution processes used by a single capture process run on a single instance in a Real Application Clusters environment.

See Also:

- ["Streams Queues and Oracle Real Application Clusters"](#) on page 3-18
- ["Streams Apply Processes and Oracle Real Application Clusters"](#) on page 4-29
- *Oracle9i Database Administrator's Guide* for more information on the ARCHIVE_LAG_TARGET initialization parameter
- *Oracle9i Database Reference* for more information about the DBA_QUEUE_TABLES data dictionary view
- ["The Persistent State of a Capture Process"](#) on page 2-30
- ["Supplemental Logging in a Streams Environment"](#) on page 2-11

Capture Process Architecture

A capture process is an Oracle background process whose process name is `cpnn`, where `nn` is a capture process number. Valid capture process names include `cp01` through `cp99`. 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 rules that control which changes a capture process captures.

The user who creates a capture process is the user who performs capture rule evaluations and capture rule-based transformations. This user also enqueues captured events into the queue used by the capture process. This user must have the necessary privileges to perform these actions, including execute privilege on the rule set used by the capture process, execute privilege on all transformation functions used in the rule set, and privileges to enqueue events into the capture process queue.

See Also: ["Configuring a Streams Administrator"](#) on page 11-2 for information about the required privileges

This section discusses the following topics:

- [Capture Process Components](#)
- [Alternate Tablespace for LogMiner Tables](#)
- [Capture Process Creation](#)
- [ARCHIVELOG Mode and a Capture Process](#)
- [Capture Process Parameters](#)
- [Capture Process Rule Evaluation](#)
- [The Persistent State of a Capture Process](#)

Capture Process Components

The components of a capture process depend on the setting specified for the `parallelism` capture process parameter. If `parallelism` is set to a value of 3 or greater, then a capture process uses the following parallel execution servers to capture changes concurrently:

- One **reader server** that reads the redo log to find changes
- A number of **preparer servers** that format changes found by the reader into LCRs. The number of preparer servers equals the number specified for the `parallelism` capture process parameter minus two.
- One **builder server** that merges the LCRs created by the preparer servers to preserve the SCN order. After merging the LCRs, the builder server enqueues them into the queue associated with the capture process.

For example, if `parallelism` is set to 5, then a capture process uses a total of five parallel execution servers, assuming five parallel execution servers are available: one reader server, three preparer servers, and one builder server.

If `parallelism` is set to 2 or lower, then a capture process itself (`cpnn`) performs all the work without using any parallel execution servers.

See Also:

- ["Capture Process Parallelism"](#) on page 2-26 for more information about the `parallelism` parameter
- *Oracle9i Database Administrator's Guide* for information about managing parallel execution servers

LogMiner Configuration

The capture process uses LogMiner to capture changes that are recorded in the redo log. This section describes configuring LogMiner for use by one or more capture processes.

Alternate Tablespace for LogMiner Tables

LogMiner tables include data dictionary tables and temporary tables used by LogMiner. By default, all LogMiner tables are created to use the `SYSTEM` tablespace, but the `SYSTEM` tablespace may not have enough space to accommodate the LogMiner tables. Therefore, Oracle Corporation strongly recommends creating an alternate tablespace for the LogMiner tables before you create a capture process at a database. Use the `DBMS_LOGMNR_D.SET_TABLESPACE` routine to re-create all LogMiner tables in an alternate tablespace.

See Also:

- ["Specifying an Alternate Tablespace for LogMiner"](#) on page 11-13
- *Oracle9i Database Administrator's Guide* for more information about using an alternate tablespace for LogMiner tables
- *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about the `SET_TABLESPACE` procedure

Multiple Capture Processes in a Single Database

Each capture process uses one LogMiner session, and the `LOGMNR_MAX_PERSISTENT_SESSIONS` initialization parameter controls the maximum number of active LogMiner sessions allowed in the instance. The default setting for this initialization parameter is 1. Therefore, to use multiple capture processes in a database, set the `LOGMNR_MAX_PERSISTENT_SESSIONS` initialization parameter to a value higher than the number of capture processes.

In addition, if you run multiple capture processes on a single database, you might need to increase the System Global Area (SGA) size for each instance. Use the `SGA_MAX_SIZE` initialization parameter to increase the SGA size. Also, you should increase the size of the shared pool by 10 MB for each capture process on a database.

Note: Oracle Corporation recommends that each capture process use a separate queue to keep LCRs from different capture processes separate.

See Also: *Oracle9i Database Reference* for more information about the `LOGMNR_MAX_PERSISTENT_SESSIONS` initialization parameter

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 are added to the rule set automatically. The `DBMS_STREAMS_ADM` package was designed for use in replication environments. Alternatively, using the `DBMS_CAPTURE_ADM` package to create a capture process is more flexible, and you create a rule set 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 the rule set for the capture process.

When a capture process is created by a procedure in the `DBMS_STREAMS_ADM` package, a procedure in the `DBMS_CAPTURE_ADM` package is run automatically on the tables whose changes will be captured by the capture process. The following table lists which procedure is run in the `DBMS_CAPTURE_ADM` package when you run a procedure in the `DBMS_STREAMS_ADM` package.

When you run this procedure in the <code>DBMS_STREAMS_ADM</code> package	This procedure in the <code>DBMS_CAPTURE_ADM</code> package is run automatically
<code>ADD_TABLE_RULES</code>	<code>PREPARE_TABLE_INSTANTIATION</code>
<code>ADD_SCHEMA_RULES</code>	<code>PREPARE_SCHEMA_INSTANTIATION</code>
<code>ADD_GLOBAL_RULES</code>	<code>PREPARE_GLOBAL_INSTANTIATION</code>

More than one call to prepare instantiation is allowed. When a capture process is created by the `CREATE_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package, you must run the appropriate procedure manually to prepare each table, schema, or database whose changes will be captured for instantiation, if you plan to instantiate the table, schema, or database at a remote site.

Note: After creating a capture process at a database, do not change the `DBID` or `DBNAME` of the database.

See Also: [Chapter 12, "Managing a Capture Process"](#) and *Oracle9i Supplied 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_TABLE_RULES`
- `DBMS_STREAMS_ADM.ADD_SCHEMA_RULES`
- `DBMS_STREAMS_ADM.ADD_GLOBAL_RULES`
- `DBMS_CAPTURE_ADM.CREATE_CAPTURE`

Data Dictionary Duplication During Capture Process Creation

When the first capture process is created for a database, Streams populates a duplicate data dictionary called a Streams data dictionary for use by capture processes and propagations. Initially, the Streams data dictionary is consistent with the primary data dictionary at the time when the capture process was created.

A capture process requires a Streams 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 or hours 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 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 the data dictionary to determine the details of the change.

The Streams data dictionary is updated when a DDL statement is processed by a capture process, if necessary. If there were any DDL changes to the relevant tables in the time between when a capture process is capturing changes and the current time, then the primary data dictionary may not contain the correct information for the captured changes. However, the Streams data dictionary always reflects the correct time for the captured changes because it versions a subset of the information in the primary data dictionary.

When a capture process determines whether or not to capture DDL changes involving a table, the capture process automatically adds information about the change to the Streams data dictionary. In addition, the capture process determines whether or not to capture the Streams data dictionary information for the new version of the table. To make these determinations, the capture rule set is evaluated with partial information that includes the name and owner of the table created or altered by the DDL statement. Capturing and propagating Streams data dictionary information makes it available in each destination queue, where it can be used by propagations and apply processes.

If at least one rule in the capture rule set either evaluates to `TRUE` (`true_rules`) or could evaluate to `TRUE` given more information (`maybe_rules`), then the Streams data dictionary information is captured for the table. This rule can be either a DML rule or a DDL rule. A capture process at a source database performs a similar rule evaluation when a table is prepared for instantiation.

Because the data dictionary is duplicated when the first capture process is created, it might take some time to create the first capture process for a database. The amount of time required depends on the number of database objects in the database.

The data dictionary is duplicated only once for a database. Additional capture processes use the same Streams data dictionary that the first capture process created in the database. Because the Streams data dictionary is multiversions, each capture process is in sync with the Streams data dictionary.

See Also:

- ["Capture Process Rule Evaluation"](#) on page 2-27
- ["Preparing Database Objects for Instantiation at a Source Database"](#) on page 12-11
- ["Streams Data Dictionary for Propagations"](#) on page 3-25
- ["Streams Data Dictionary for an Apply Process"](#) on page 4-33

Scenario Illustrating the Need for a Streams 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 Streams data dictionary contain different information. [Table 2-1](#) illustrates how the information in the Streams 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 Capture Data Dictionaries

Current Time	Change Capturing Time	Primary Data Dictionary	Streams Data Dictionary
1	1	Table <i>t1</i> has columns <i>a</i> and <i>b</i> .	Table <i>t1</i> has columns <i>a</i> and <i>b</i> at time 1.
2	1	Table <i>t1</i> has columns <i>a</i> , <i>b</i> , and <i>c</i> .	Table <i>t1</i> has columns <i>a</i> and <i>b</i> at time 1.
3	1	Table <i>t1</i> has columns <i>a</i> and <i>c</i> .	Table <i>t1</i> has columns <i>a</i> and <i>b</i> at time 1.

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 was actually inserted into column *b*.

Because the capture process uses the Streams data dictionary, the error is avoided. The Streams 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`.

ARCHIVELOG Mode and a Capture Process

A capture process reads online redo logs whenever possible and archived redo logs otherwise. For this reason, the 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 ever need that file. Make sure redo logs are available until all transactions within the redo log have been applied at all downstream databases. You can use `APPLIED_SCN` column in the `DBA_CAPTURE` data dictionary view to determine the SCN of the most recent message dequeued by the relevant apply processes. All changes below this SCN have been dequeued by all apply processes that apply changes captured by the capture process.

When a capture process falls behind, there is a seamless transition from reading an online redo log to reading an archived redo log, and, when a capture process catches up, there is a seamless transition from reading an archived redo log to reading an online redo log.

See Also: *Oracle9i Database Administrator's Guide* for information about running a database in ARCHIVELOG mode

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. After you set the capture process parameters, you can start the capture process.

See Also:

- ["Setting a Capture Process Parameter"](#) on page 12-8
- This section does not discuss all of the available capture process parameters. See the `DBMS_CAPTURE_ADM.SET_PARAMETER` procedure in the *Oracle9i Supplied 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 processes. The preparer servers concurrently format changes found in the redo log into LCRs.

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 2-19 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 (`cpnn`) remains the same.

Capture Process Rule Evaluation

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 set 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 change is converted into an LCR if one or more rules may evaluate to `TRUE` after conversion.
 - A change is not converted into an LCR if the capture process can ensure that no rules would evaluate to `TRUE` after conversion.
3. Converts changes that may cause one or more rules to evaluate to `TRUE` 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 based on the rules.
 6. Enqueues the remaining captured LCRs into the queue associated with the capture process.

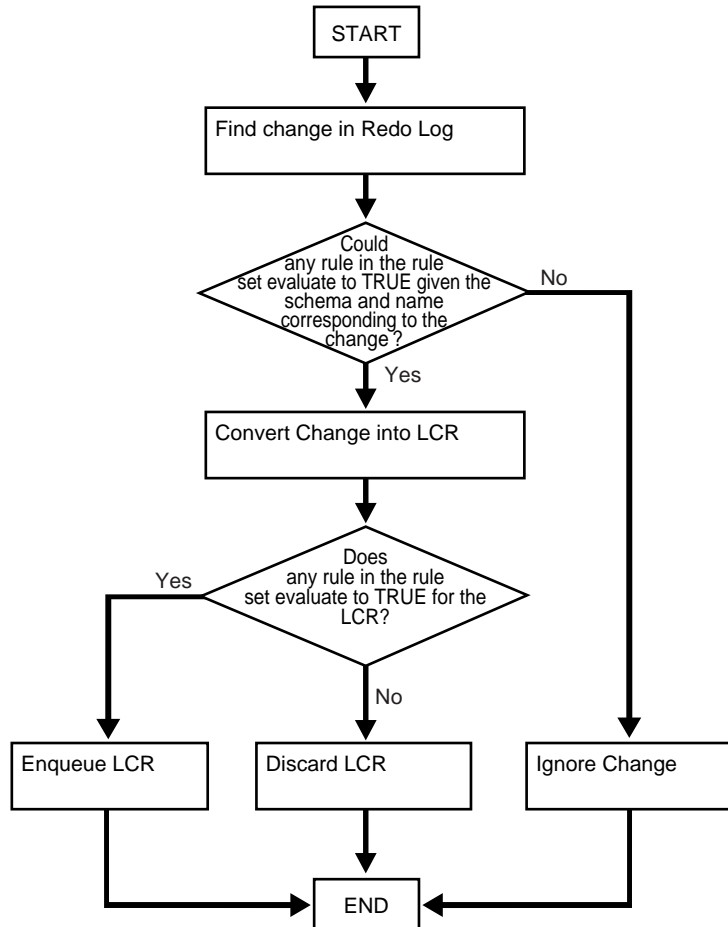
For example, suppose the following rule is defined for a capture process: Capture changes to the `hr.employees` table where the `department_id` is 50. No other rules are defined for the capture process, and the `parallelism` parameter for the capture process is set to 1.

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.

Figure 2-2 illustrates capture process rule evaluation in a flowchart.

Figure 2-2 Flowchart Showing Capture Process Rule Evaluation



The Persistent State of a Capture Process

A capture process maintains a persistent state. That is, the capture process maintains its current state when the database is shut down and restarted. For example, if the capture process is running when the database is shut down, then the capture process automatically starts when the database is restarted, but, if the capture process is stopped when a database is shut down, then the capture process remains stopped when the database is restarted.

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:

- [Event Staging and Propagation Overview](#)
- [Captured and User-Enqueued Events](#)
- [Event Propagation Between Queues](#)
- [SYS.AnyData Queues and User Messages](#)
- [Streams Queues and Oracle Real Application Clusters](#)
- [Streams Staging and Propagation Architecture](#)

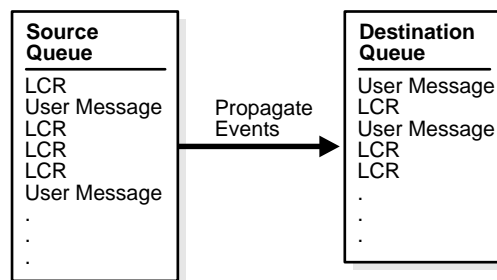
See Also: [Chapter 13, "Managing Staging and Propagation"](#)

Event Staging and Propagation Overview

Streams uses queues of type `SYS.AnyData` to stage events. There are two types of events that can be staged in a Streams queue: logical change records (LCRs) and user messages. LCRs are objects that contain information about a change to a database object, while user messages are custom messages created by users or applications. Both types of events are of type `SYS.AnyData` and can be used for information sharing within a single database or between databases.

Staged events can be consumed or propagated, or both. These events can be consumed by an apply process or by a user application that explicitly dequeues them. Even after an event is consumed, it may remain in the queue if you have also configured Streams to propagate the event to one or more other queues or if message retention is specified. These other queues may reside in the same database or in different databases. 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. This user must have the necessary privileges to propagate events. These privileges include the following:

- Execute privilege on the rule set used by the propagation
- Execute privilege on all transformation functions used in the rule set
- 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.

See Also: *Oracle9i Application Developer's Guide - Advanced Queuing* for more information about message retention

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 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 can either 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.
- A user application explicitly dequeues user-enqueued events and processes them. Captured events cannot be dequeued by a user application; they 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
- *Oracle9i Application Developer's Guide - Advanced Queuing* for information about enqueueing events into a queue
- [Chapter 4, "Streams Apply Process"](#) for more information about the apply process
- ["Managing Logical Change Records \(LCRs\)"](#) on page 16-2

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 the 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.

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 the 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 13-7
- *Oracle9i Application Developer's Guide - Advanced Queuing* for detailed information about the propagation infrastructure in AQ
- [Chapter 8, "Streams Tags"](#)

Propagation Rules

A propagation propagates events based on rules that you define. For LCR events, each rule specifies the database objects for which the propagation propagates changes and the types of changes to propagate. You can specify propagation rules for LCR events at the following levels:

- A table rule propagates either DML or DDL changes to a particular table.
- A schema rule propagates either DML or DDL changes to the database objects in a particular schema.
- A global rule propagates either all DML 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 Streams 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 indefinitely. 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 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. Then, you can use one source queue for the slower destination databases and another source queue for the faster destination databases.

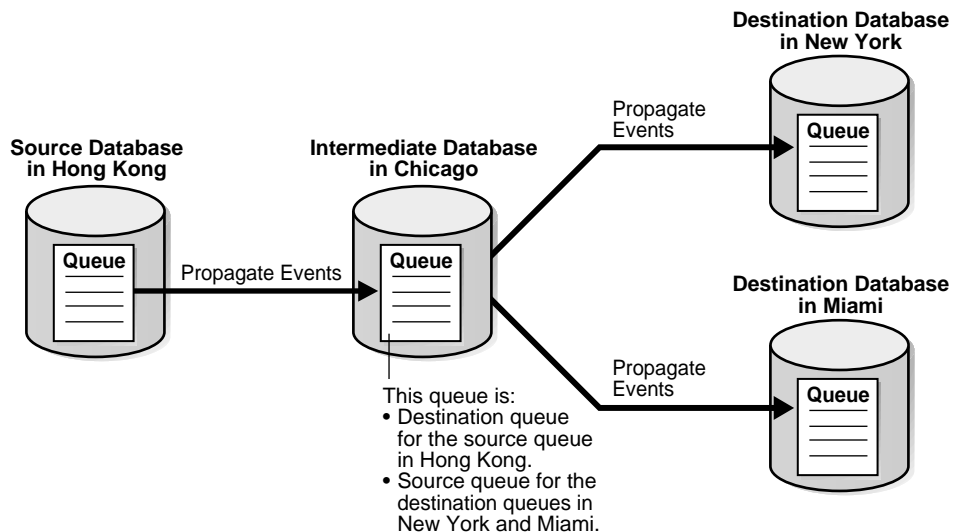
See Also:

- [Chapter 2, "Streams Capture Process"](#)
- ["Monitoring a Streams Queue"](#) on page 17-12

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



The advantage of using a directed network is that a source database need not have a physical network connection with the 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 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 source database 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.
- In a multiple source Streams environment, you can add databases to the Streams environment without stopping all DML on the objects at each database and waiting for all LCRs involving the objects to be captured, propagated, and applied. A new database is instantiated from the one database that will connect it to the rest of the Streams environment. In contrast, in a queue forwarding environment, no single database contains all of the current data for a shared object with multiple sources, and so DML should be stopped when adding new databases to the environment.

See Also:

- [Chapter 4, "Streams Apply Process"](#)
- [Chapter 22, "Single Source Heterogeneous Replication Example"](#) for an example of an environment that uses queue forwarding
- ["Primary Database Sharing Data with Several Secondary Databases"](#) on page 8-12 for an example of an environment that uses apply forwarding

SYS.AnyData Queues and User Messages

Streams enables messaging with queues of type `SYS.AnyData`. These queues are called Streams queues. Streams 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. A queue that can stage messages of only a particular type are called typed queues.

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 dequeue API 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 interoperates with 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.

See Also:

- ["Managing a Streams Messaging Environment"](#) on page 13-18
- [Chapter 19, "Streams Messaging Example"](#)
- *Oracle9i Application Developer's Guide - Advanced Queuing* for more information about AQ

SYS.AnyData Wrapper for User Messages Payloads

You can wrap almost any type of payload in a `SYS.AnyData` payload. To do this, you use the `Convertdata_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.

The following datatypes cannot be wrapped in a `SYS.AnyData` wrapper:

- Nested table
- NCLOB
- ROWID and UROWID

The following datatypes can be directly wrapped in a `SYS.AnyData` wrapper, but these datatypes cannot be present in a user-defined type payload wrapped in a `SYS.AnyData` wrapper:

- CLOB
- BLOB
- BFILE
- VARRAY

See Also:

- ["Wrapping User Message Payloads in a SYS.AnyData Wrapper" on page 13-19](#)
- *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about the `SYS.AnyData` type

Programmatic Environments for Enqueue and Dequeue of User Messages

Your applications can use the following programmatic environments to enqueue user messages into a Streams queue and dequeue user messages from a Streams queue:

- PL/SQL (DBMS_AQ package)
- JMS
- OCI

The following sections provide information about using these interfaces to enqueue user messages into and dequeue user messages from a Streams queue.

See Also: *Oracle9i Application Developer's Guide - Advanced Queuing* for more information about these programmatic interfaces

Enqueuing User Messages Using PL/SQL

To enqueue a user message containing an LCR into a Streams queue using PL/SQL, first create the LCR to be enqueued. You use the constructor for the `SYS.LCR$_ROW_RECORD` type to create a row LCR, and you use the constructor for the `SYS.LCR$_DDL_RECORD` type to create a DDL LCR. Then you use the `SYS.AnyData.ConvertObject` function to convert the LCR into `SYS.AnyData` payload and enqueue it using the `DBMS_AQ.ENQUEUE` procedure.

To enqueue a user message containing a non-LCR object into a Streams queue using PL/SQL, you use one of the `SYS.AnyData.Convert*` functions to convert the object into `SYS.AnyData` payload and enqueue it using the `DBMS_AQ.ENQUEUE` procedure.

See Also:

- ["Managing a Streams Messaging Environment"](#) on page 13-18
- [Chapter 19, "Streams Messaging Example"](#)

Enqueuing User Messages Using OCI or JMS

To enqueue a user message containing an LCR into a Streams queue using JMS or OCI, you must represent the LCR in XML format. To construct an LCR, use the `oracle.xdb.XMLType` class. LCRs are defined in the `SYS` schema. The LCR schema must be loaded into the `SYS` schema using the `catxldr.sql` script in Oracle home in the `rdbms/admin/` directory.

To enqueue a message using OCI, perform the same actions that you would to enqueue a message into a typed queue. A typed queue is a queue that can stage messages of a particular type only. To enqueue a message using JMS, a user must have `EXECUTE` privilege on `DBMS_AQ`, `DBMS_AQIN`, and `DBMS_AQJMS` packages.

A non-LCR user message can be a message of any user-defined type or a JMS type. The JMS types include the following:

- `javax.jms.TextMessage`
- `javax.jms.MapMessage`
- `javax.jms.StreamMessage`
- `javax.jms.ObjectMessage`
- `javax.jms.BytesMessage`

When using user-defined types, you must generate the Java class for the message using `Jpublisher`, which implements the `ORADData` interface. To enqueue a message into a Streams queue, you can use methods `QueueSender.send` or `TopicPublisher.publish`.

See Also:

- ["Enqueue and Dequeue Events Using JMS"](#) on page 19-35
- *Oracle9i Application Developer's Guide - Advanced Queuing* and *Oracle9i XML Database Developer's Guide - Oracle XML DB* for more information about representing messages in XML format
- *Oracle9i Supplied Java Packages Reference* for more information about the `oracle.jms` Java package
- The `OCI AQ enqueue` function in the *Oracle Call Interface Programmer's Guide* for more information about enqueueing messages using OCI

Dequeuing User Messages Using PL/SQL

To dequeue a user message from Streams queue using PL/SQL, you use the `DBMS_AQ.DEQUEUE` procedure and specify `SYS.AnyData` as the payload. The user message may contain an LCR or another type of object.

See Also:

- ["Managing a Streams Messaging Environment"](#) on page 13-18
- [Chapter 19, "Streams Messaging Example"](#)

Dequeuing User Messages Using OCI or JMS

In a Streams queue, user messages containing LCRs in XML format are represented as `oracle.xdb.XMLType`. Non-LCR messages can be one of the following formats:

- A JMS type (`javax.jms.TextMessage`, `javax.jms.MapMessage`, `javax.jms.StreamMessage`, `javax.jms.ObjectMessage`, or `javax.jms.BytesMessage`)
- A user-defined type

To dequeue a message from a Streams queue using JMS, you can use methods `QueueReceiver`, `TopicSubscriber`, or `TopicReceiver`. Because the queue may contain different types of objects wrapped in a `SYS.AnyData` wrapper, you must register a list of SQL types and their corresponding Java classes in the `typemap` of the JMS session. JMS types are already preregistered in the `typemap`.

For example, suppose a queue contains LCR messages represented as `oracle.xdb.XMLType` and messages of type `person` and `address`. The classes `JPerson.java` and `JAddress.java` are the ORAData mappings for `person` and `address`, respectively. Before dequeuing the message, the type map must be populated as follows:

```
java.util.Dictionary map = ((AQJmsSession)q_sess).getTypeMap();

map.put("SCOTT.PERSON", Class.forName("JPerson"));
map.put("SCOTT.ADDRESS", Class.forName("JAddress"));
map.put("SYS.XMLTYPE", Class.forName("oracle.xdb.XMLType")); // For LCRs
```

When using message selectors with `QueueReceiver` or `TopicPublisher`, the selector can contain any SQL92 expression that has a combination of one or more of the following:

- JMS Message header fields or properties, including `JMSPriority`, `JMSCorrelationID`, `JMSType`, `JMSXUserI`, `JMSXAppID`, `JMSXGroupID`, and `JMSXGroupSeq`. The following is an example of a JMS message field:

```
JMSPriority < 3 AND JMSCorrelationID = 'Fiction'
```

- User-defined message properties, as in the following example:

```
color IN ('RED', 'BLUE', 'GREEN') AND price < 30000
```

- PL/SQL functions, as in the following example:

```
hr.GET_TYPE(tab.user_data) = 'HR.EMPLOYEES'
```

To dequeue a message using OCI, perform the same actions that you would to dequeue a message from a typed queue.

See Also:

- ["Enqueue and Dequeue Events Using JMS"](#) on page 19-35
- *Oracle9i Application Developer's Guide - Advanced Queuing and Oracle9i XML Database Developer's Guide - Oracle XML DB* for more information about representing messages in XML format
- *Oracle9i Supplied Java Packages Reference* for more information about the `oracle.jms` Java package
- The `OCIAQdeq` function in the *Oracle Call Interface Programmer's Guide* for more information about dequeuing messages using OCI

Message Propagation and SYS.AnyData Queues

SYS.AnyData queues can interoperate with typed queues in a Streams environment. A typed queue can stage messages of a particular type only. [Table 3-1](#) shows the types of propagation possible between queues.

Table 3-1 Propagation Between Different Types of Queues

Source Queue	Destination Queue	Transformation
SYS.AnyData	SYS.AnyData	None
Typed	SYS.AnyData	Implicit Note: Propagation is possible only if the messages in the typed queue meet the restrictions outlined in " User-Defined Type Messages " on page 3-17.
SYS.AnyData	Typed	Requires a rule to filter messages and a user-defined transformation
Typed	Typed	Follows Advanced Queuing (AQ) rules (see <i>Oracle9i Application Developer's Guide - Advanced Queuing</i> for information)

To propagate messages containing a payload of a certain type from a SYS.AnyData source queue to a typed destination queue, you must perform a transformation. Only messages containing a payload of the same type as the typed queue can be propagated to the typed queue.

Although you cannot use Simple Object Access Protocol (SOAP) to interact directly with a Streams queue, you can use SOAP with Streams by propagating messages between a Streams queue and a typed queue. If you want to enqueue a message into a Streams queue using SOAP, then you can configure propagation from a typed queue to Streams queue. Then, you can use SOAP to enqueue a message into the typed queue. The message will be propagated automatically from the typed queue to the Streams queue.

If you want to use SOAP to dequeue a message that is in a Streams queue, then you can configure propagation from a Streams queue to a typed queue. The message will be propagated automatically from the Streams queue to the typed queue. Then, the message would be available for access using SOAP.

Note: Certain Streams capabilities, such as capturing changes using a capture process and applying changes with an apply process, can be configured only with `SYS.AnyData` queues.

See Also:

- ["Propagating Messages Between a SYS.AnyData Queue and a Typed Queue"](#) on page 13-24
- *Oracle9i Application Developer's Guide - Advanced Queuing* for more information about using SOAP and typed queues

User-Defined Type Messages

If you plan to enqueue, propagate, or dequeue user-defined type messages in a Streams environment, then each type used in these messages must exist at every database where the message may be staged in a queue. Some environments use directed networks to route messages through intermediate databases before they reach their destination. In such environments, the type must exist at each intermediate database, even if the messages of this type are never enqueued or dequeued at a particular intermediate database.

In addition, the following requirements must be met for such types:

- The type name must be the same at each database.
- The type must be in the same schema at each database.
- The shape of the type must match exactly at each database.
- The type cannot use inheritance or type evolution at any database.
- The type cannot contain varrays, nested tables, LOBs, rowids, or urowids.

The object identifier (OID) need not match at each database.

See Also:

- ["SYS.AnyData Wrapper for User Messages Payloads"](#) on page 3-11 for information about wrapping user-defined type message payloads in `SYS.AnyData` messages
- ["Directed Networks"](#) on page 3-7

Streams Queues and Oracle Real Application Clusters

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

A Streams queue that contains only user-enqueued events behaves the same as a typed queue in a Real Application Clusters environment. However, if a Streams queue contains or will contain captured events in a Real Application Clusters environment, then the environment must meet the following requirements:

- Each queue table containing a Streams queue with captured events must be created using the `SET_UP_QUEUE` procedure in the `DBMS_STREAMS_ADM` package. Creating or altering a queue table with the `DBMS_AQADM` package is not supported if any queue in the queue table contains captured events.
- All capture processes and apply processes that deal with captured events and use a particular Streams queue must be started on the owner instance for the queue.
- Each propagation that propagates captured events to a Real Application Clusters 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.
- The AQ time manager must be running on all instances. Therefore, the `AQ_TM_PROCESSES` initialization parameter must be set to at least 1 on each 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. 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 2-17
- ["Streams Apply Processes and Oracle Real Application Clusters"](#) on page 4-29
- ["Queue Buffers"](#) on page 3-19
- *Oracle9i Database Reference* for more information about the `DBA_QUEUE_TABLES` data dictionary view
- *Oracle9i Application Developer's Guide - Advanced Queuing* for more information about queues and Real Application Clusters

Streams Staging and Propagation Architecture

In general, Streams queues and propagations use the infrastructure of AQ. However, unlike an AQ queue, which stages all events in a queue table, a Streams queue has a queue buffer to stage captured events in shared memory. This section describes queue buffers and discusses how queue buffers are used in a Real Application Clusters environment. This section also discusses 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.

See Also:

- *Oracle9i Application Developer's Guide - Advanced Queuing* for more information about AQ infrastructure
- *Oracle9i Database Administrator's Guide* for more information about the `DBMS_JOB` package

Queue Buffers

A **queue buffer** is System Global Area (SGA) memory associated with a Streams queue that contains only captured events. A queue buffer 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 Streams queue. Such a database may be a source database, an intermediate database, or a destination database. User-enqueued LCR events and user-enqueued non-LCR events are always staged in queue tables, not in queue buffers.

Queue buffers improve performance, but the contents of a queue buffer are lost if the instance containing the buffer shuts down normally or abnormally. Streams automatically recovers from these cases, assuming full database recovery is performed on the instance.

In a single database, all of the queue buffers combined can use up to 10% of SGA memory. A queue buffer may overflow if there is not enough shared memory available to hold captured events. Captured events that overflow a queue buffer 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. If the events in a queue buffer are lost, the events spilled from the queue buffer are subsequently deleted in order to keep the queue buffer and its queue table in sync. Also, when a transaction is moved to an exception queue, all events in the transaction are staged in a queue table, not in a queue buffer.

See Also:

- *Oracle9i Database Concepts* for more information about the SGA
- ["Performing Database Point-in-Time Recovery on a Destination Database"](#) on page 16-29

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 (`Jnnn`) 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:

- Currently, 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_RULE` procedure in the `DBMS_STREAMS_ADM` package
- The `ADD_SCHEMA_PROPAGATION_RULE` procedure in the `DBMS_STREAMS_ADM` package
- The `ADD_TABLE_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 five seconds, which is the wait time for a message to be propagated to a destination queue after it is enqueued into a queue with no messages requiring propagation to the same destination queue.

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 13-12

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 explicitly associated 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 processes enqueue events into a queue and dequeue events from a queue.

All Streams 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_AQ` package. The `SET_UP_QUEUE` procedure does not grant this privilege.

To configure the 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 it is used. `SET_UP_QUEUE` then runs the `ENABLE_DB_ACCESS` procedure in the `DBMS_AQADM` package, specifying the agent and the user. 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`, then an error is raised. 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. Then, retry `SET_UP_QUEUE`.

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.

For a capture process, the user who invokes the procedure that creates the capture process 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 `apply_user` is specified, then the user who invokes the procedure that creates the apply process is the user who runs the apply process.

Also, if you change the `apply_user` for an apply process using the `ALTER_APPLY` procedure in the `DBMS_APPLY_ADM` package, then the specified `apply_user` is configured as a secure queue user of the queue used by the apply process. However, the old apply user remains configured as a secure queue user of the queue. To remove the old apply user, run the `DISABLE_DB_ACCESS` procedure in the `DBMS_AQADM` package, specifying the old apply 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.

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.

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 drop a capture process or an apply process, then the users who were configured as secure queue users for these processes 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.

See Also:

- ["Enabling a User to Perform Operations on a Secure Queue"](#) on page 13-3
- ["Disabling a User from Performing Operations on a Secure Queue"](#) on page 13-5
- *Oracle9i Supplied 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 3-2](#) shows apply process behavior for each type of event and each type of queue.

Table 3-2 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 Streams Queues"](#) on page 13-2
- *Oracle9i Application Developer's Guide - Advanced Queuing for* more information about message grouping

Streams Data Dictionary for Propagations

When a capture process is created, a duplicate data dictionary called the Streams data dictionary is populated automatically. The Streams data dictionary is a multiversed 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 when a capture process evaluates rules and creates LCRs. This mapping keeps each captured event as small as possible because the event can store numbers rather than names.

The mapping information in the Streams data dictionary at the source database may be 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 multiversed Streams data dictionary at each site 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 set for the propagation. When a propagation encounters Streams data dictionary information for a table, the propagation rule set is evaluated with partial information that includes the source database name, table name, and table owner.

If at least one rule in the rule set either evaluates to `TRUE` (`true_rules`) or could evaluate to `TRUE` given more information (`maybe_rules`), then the Streams data dictionary information is propagated. This rule can be either a DML rule or a DDL rule.

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.

See Also:

- ["Data Dictionary Duplication During Capture Process Creation"](#)
on page 2-22
- [Chapter 6, "How Rules Are Used In Streams"](#)

Streams Apply Process

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

This chapter contains these topics:

- [Apply Process Overview](#)
- [Apply Rules](#)
- [Event Processing with an Apply Process](#)
- [Datatypes Applied](#)
- [Considerations for Applying DML Changes to Tables](#)
- [Considerations for Applying DDL Changes](#)
- [Trigger Firing Property](#)
- [Instantiation SCN and Ignore SCN](#)
- [The Oldest SCN for an Apply Process](#)
- [Low-Watermark and High-Watermark for an Apply Process](#)
- [Streams Apply Processes and RESTRICTED SESSION](#)
- [Streams Apply Processes and Oracle Real Application Clusters](#)
- [Apply Process Architecture](#)

See Also: [Chapter 14, "Managing an Apply Process"](#)

Apply Process Overview

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 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-defined message dequeued by an apply process is of type `SYS.AnyData` and can contain any user message, including a user-created LCR.

Events applied by an apply process are applied by an **apply user**. The apply user is the user who applies all DML statements and DDL statements and who runs user-defined apply handlers.

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

Apply Rules

An apply process applies changes based on rules that you define. Each rule specifies the database objects to which an apply process applies changes and the types of changes to apply. You can specify apply rules at the following levels:

- A table rule applies either DML or DDL changes to a particular table. Subset rules are table rules that include a subset of the changes to a particular table.
- A schema rule applies either DML or DDL changes to the database objects in a particular schema.
- A global rule applies either all DML 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 a procedure in the `DBMS_STREAMS_ADM` package, the apply process applies only captured 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.

The database where an event originated is important to an apply process for captured events but not for user-enqueued events. For a captured event, the source database is the database where the change was generated in the redo log. An apply process must determine the source database for each captured LCR to ensure that it applies changes from only one source database, and a database administrator must ensure that an apply process applies changes from only one capture process at that source database. For a user-enqueued event, an apply process ignores information about the database where the event originated, even if the event is a user-enqueued LCR. A single apply process can apply user-enqueued events that originated at different databases.

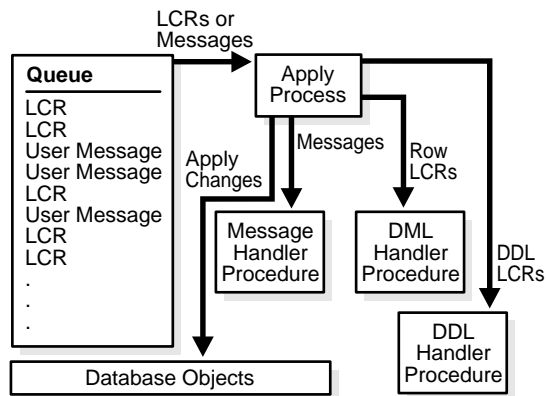
See Also:

- ["Event Staging and Propagation Overview"](#) on page 3-2 for more information about captured and user-enqueued events
- *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about the `CREATE_APPLY` procedure

Event Processing Options

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

Each apply process can apply captured events from only one source database, because processing the LCRs in these events requires knowledge of the dependencies, meaningful transaction ordering, and transactional boundaries at the source database. Captured LCRs from multiple databases may be sent to a single destination queue. However, if a single queue contains captured LCRs from multiple 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 events 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 Streams queue used by a capture process or apply process have captured events 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 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 error handler cannot resolve the error, then the apply process places the transaction, and all LCRs associated with the transaction, into an exception 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 then 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 savepoint that the user procedure has established. To execute DDL inside a DDL handler, invoke the `EXECUTE` member procedure for the LCR.

To set a DML handler, use the `SET_DML_HANDLER` procedure in the `DBMS_APPLY_ADM` package, and this setting is used by all apply processes in the database. 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. Then, the 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.

Note: When you run the `SET_DML_HANDLER` procedure, you specify the object for which the handler is used, and Oracle checks to ensure that the specified object exists in the local destination database. If the object does not exist, then an error is raised. Therefore, if the name of the object is different at the source database and destination database, then use a rule-based transformation to convert the object name in the row LCR before the row LCR is applied.

See Also:

- ["Logical Change Records \(LCRs\)"](#) on page 2-2 for more information about row LCRs and DDL LCRs
- *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about the `EXECUTE` member procedure for LCR types
- ["Rule-Based Transformations"](#) on page 6-25

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, if the user-enqueued event satisfies at least one rule in the rule set for the 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. Therefore, if dependencies exist between these messages in your environment, then Oracle Corporation recommends that you set apply process parallelism to 1.

Summary of Event Processing Options

[Table 4-1](#) summarizes the event processing options available when you are using one or more of the event handlers described in the previous sections. Event 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 at least one of the rules in the rule set for the apply process.

Table 4-1 Summary of Event Processing Options

Type of Event	Default Apply Process Behavior	User Procedure	Scope of User Procedure
Row LCR	Execute DML	DML Handler or Error Handler	One operation on one table
DDL LCR	Execute DDL	DDL Handler	Entire apply process
Non-LCR User Message	Create error transaction (if no message handler exists)	Message Handler	Entire apply process

Note:

- Apply 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.
-
-

See Also:

- *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about the `EXECUTE` member procedure for LCR types
- [Chapter 8, "Streams Tags"](#)

Datatypes Applied

When applying row LCRs for data manipulation language (DML) changes to tables, an apply process applies changes made to columns of the following datatypes:

- CHAR
- VARCHAR2
- NCHAR
- NVARCHAR2
- NUMBER
- DATE
- CLOB
- BLOB
- RAW
- TIMESTAMP
- TIMESTAMP WITH TIME ZONE
- TIMESTAMP WITH LOCAL TIME ZONE
- INTERVAL YEAR TO MONTH
- INTERVAL DAY TO SECOND

The apply process does not apply DML changes in columns of the following datatypes: NCLOB, LONG, LONG RAW, BFILE, ROWID, and UROWID, and user-defined type (including object types, REFS, varrays, and nested tables). The apply process raises an error if it attempts to apply a row LCR that contains information about a column of an unsupported datatype. Then, the apply process moves the transaction that includes the LCR into an exception queue.

See Also:

- ["Datatypes Captured"](#) on page 2-6
- *Oracle9i SQL Reference* for more information about these datatypes

Considerations for Applying DML Changes to Tables

The following sections discuss considerations for applying DML changes to tables:

- [Constraints](#)
- [Substitute Key Columns](#)
- [Row Subsetting Using Streams Rules](#)
- [Apply Process Behavior for Column Discrepancies](#)
- [Conflict Resolution and an Apply Process](#)
- [Handlers and Row LCR Processing](#)

Constraints

You must ensure that the primary key columns at the destination database are logged in the redo log at the source database for every update. A unique or foreign key constraint at a destination database that contains data from more than one column at the source database requires additional logging at the source database.

There are various ways to ensure that a column is logged at the source database. For example, whenever the value of a column is updated, the column is logged. Also, Oracle has a feature called supplemental logging that automates the logging of specified columns.

For a unique key and foreign key constraint at a destination database that contains data from only one column at a source database, no supplemental logging is required. However, for a constraint that contains data from multiple columns at the source database, you must create a conditional supplemental log group containing all the columns at the source database that are used by the constraint at the destination database.

Typically, unique key and foreign key constraints include the same columns at the source database and destination database. However, in some cases, an apply handler or rule-based transformation may combine a multi-column constraint from the source database into a single key column at the destination database. Also, an apply handler or rule-based transformation may separate a single key column from the source database into a multi-column constraint at the destination database. In such cases, the number of columns in the constraint at the source database determines whether a conditional supplemental log group is required. If there is more than one column in the constraint at the source database, then a conditional supplemental log group containing all the constraint columns is required at the

source database. If there is only one column in the constraint at the source database, then no supplemental logging is required for the key column.

See Also: ["Supplemental Logging in a Streams Environment"](#) on page 2-11

Substitute Key Columns

If possible, each table for which changes are applied by an apply process should have a primary key. When a primary key is not possible, Oracle Corporation recommends that each table have a set of columns that can be used as a unique identifier for each row of the table. If the tables that you plan to use in your Streams environment do not have a primary key or a set of unique columns, then consider altering these tables accordingly.

To detect conflicts and handle errors accurately, Oracle must be able to identify uniquely and match corresponding rows at different databases. By default, Streams uses the primary key of a table to identify rows in the table. When a table at a destination database does not have a primary key, or when you want to use columns other than the primary key for the key, you can designate a substitute key at the destination database. A substitute key is a column or set of columns that Oracle can use to identify rows in the table during apply.

You can specify the substitute primary key for a table using the `SET_KEY_COLUMNS` procedure in the `DBMS_APPLY_ADM` package. Unlike true primary keys, the substitute key columns may contain `NULLS`. Also, the substitute key columns take precedence over any existing primary key for the specified table for all apply processes at the destination database.

If you specify a substitute key for a table in a destination database, and these columns are not a primary key for the same table at the source database, then you must create an unconditional supplemental log group containing the substitute key columns at the source database.

In the absence of both substitute key columns and a primary key constraint, an apply process uses all of the columns in the table as the key columns, excluding `LOB` and `LONG` columns. In this case, you must create an unconditional supplemental log group containing these columns at the source database. Using substitute key columns is preferable when there is no primary key constraint for a table because fewer columns are needed in the row LCR.

Note:

- Oracle Corporation recommends that each column you specify as a substitute key column be a NOT NULL column. You should also create a single index that includes all of the columns in a substitute key. Following these guidelines improves performance for updates, deletes, and piecewise updates to LOBs because the database can locate the relevant row more efficiently.
 - You should not permit applications to update the primary key or substitute key columns of a table. This ensures that the database can identify rows and preserve the integrity of the data.
-
-

See Also:

- The `DBMS_APPLY_ADM.SET_KEY_COLUMNS` procedure in the *Oracle9i Supplied PL/SQL Packages and Types Reference*
- ["Supplemental Logging in a Streams Environment"](#) on page 2-11

Row Subsetting Using Streams Rules

You can use the `ADD_SUBSET_RULES` procedure in the `DBMS_STREAMS_ADM` package to maintain a subset of the rows in a particular table. To do this, you specify a condition similar to the condition in the `WHERE` clause of a `SELECT` statement in the `dml_condition` parameter for this procedure. For a particular table, only one subset rule is allowed for a particular apply process.

Note: Creating subset rules for tables that have one or more LOB columns is not supported.

See Also: ["Table and Subset Rules"](#) on page 6-6

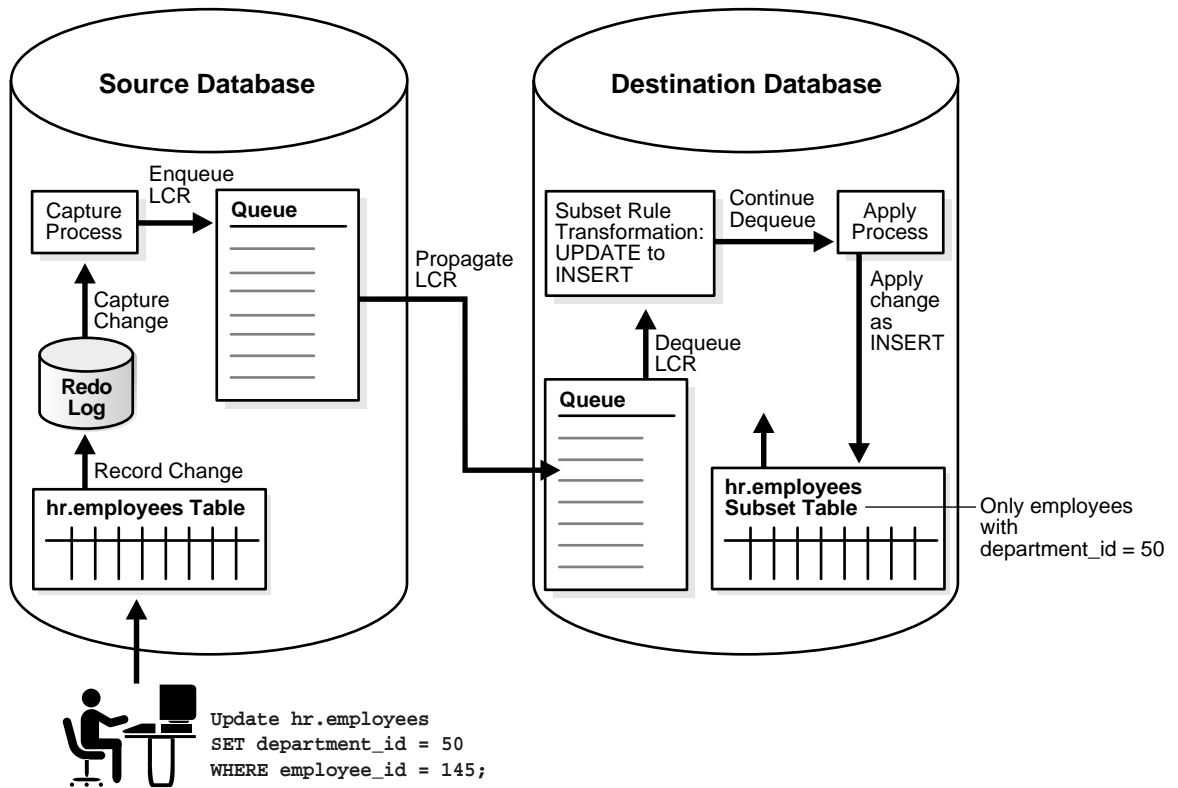
Row Migration

When you use subset rules, a captured update operation may be converted to an insert or delete operation when it is applied by an apply process. This automatic

conversion is called **row migration** and is performed by an internal LCR transformation specified in a rule's action context.

For example, suppose you use a subset rule to specify that an apply process applies changes only to the `hr.employees` table when 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 4–2 Subset Rule Transformation Example



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.

If an apply process may perform row migration when applying changes to a table and 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. To avoid such errors, Oracle Corporation recommends that you ensure that all DML performed on a subset table satisfy the subset condition.

Supplemental Logging and Row Subsetting

If you specify a subset rule for a table at a destination database, then an unconditional supplemental log group must be specified at the source database for all of the columns in the table at the destination database and all the columns in the subset condition. In certain cases, when a subset rule is specified, an update may be converted to an insert by an apply process, and in these cases supplemental information may be needed for some or all of the columns.

For example, if you specify a subset rule 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: ["Supplemental Logging in a Streams Environment"](#) on page 2-11

Apply Process Behavior for Column Discrepancies

A column discrepancy is any difference in the columns in a table at a source database and the columns in the same table at a destination database. If there are column discrepancies in your Streams environment, then use rule-based transformations or DML handlers to make the columns in row LCRs being applied by an apply process match the columns in the relevant tables at a destination database. The following sections describe apply process behavior for common column discrepancies.

See Also:

- ["Rule-Based Transformations"](#) on page 6-25 and ["Managing Rule-Based Transformations"](#) on page 15-11
- ["LCR Event Processing"](#) on page 4-4 for more information about apply process handlers
- *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about LCRs

Missing Columns at the Destination Database

If the table at the destination database is missing one or more columns that are in the table at the source database, then an apply process raises an error and moves the transaction that caused the error into an exception queue. You can avoid such an error by creating a rule-based transformation or DML handler that eliminates the missing columns from the LCRs before they are applied. Specifically, the transformation or handler can remove the extra columns using the `DELETE_COLUMN` member procedure on the row LCR.

Extra Columns at the Destination Database

If the table at the destination database has more columns than the table at the source database, then apply process behavior depends on whether the extra columns are required for dependency computations. If the extra columns are not used for dependency computations, then an apply process applies changes to the destination table. In this case, if column defaults exist for the extra columns at the destination database, then these defaults are used for these columns for all inserts. Otherwise, these inserted columns are `NULL`.

If, however, the extra columns are used for dependency computations, then an apply process places the transactions that include these changes in an exception queue. The following types of columns are required for dependency computations:

- For all changes, all key columns
- For `INSERT` and `DELETE` statements, all columns involved with constraints
- For `UPDATE` statements, if a constraint column is changed, such as a unique key constraint column or a foreign key constraint column, then all columns involved in the constraint

Column Datatype Mismatch

If the datatype for a column in a table at the destination database does not match the datatype for the same column at the source database, then an apply process places transactions containing the changes to the mismatched column into an exception queue. To avoid such an error, you can create a rule-based transformation or DML handler that converts the datatype.

Conflict Resolution and an Apply Process

Conflicts are possible in a Streams configuration where data is shared between multiple databases. A conflict can occur if DML changes are allowed to a table for which changes are captured and to a table where these changes are applied.

For example, a transaction at the source database may update a row at nearly the same time as a different transaction that updates the same row at a destination database. In this case, if data consistency between the two databases is important, then when the change is propagated to the destination database, an apply process must be instructed either to keep the change at the destination database or replace it with the change from the source database. When data conflicts occur, you need a mechanism to ensure that the conflict is resolved in accordance with your business rules.

Streams automatically detects conflicts and, for update conflicts, tries to use an update conflict handler to resolve them if one is configured. Streams offers a variety of prebuilt handlers that enable you to define a conflict resolution system for your database that resolves conflicts in accordance with your business rules. If you have a unique situation that a prebuilt conflict resolution handlers cannot resolve, then you can build and use your own custom conflict resolution handlers in an error handler or DML handler.

See Also: [Chapter 7, "Streams Conflict Resolution"](#)

Handlers and Row LCR Processing

Any of the following handlers may process a row LCR:

- DML handler
- Error handler
- Update conflict handler

The following sections describe the possible scenarios involving these handlers:

- [No Relevant Handlers](#)
- [Relevant Update Conflict Handler](#)
- [DML Handler But No Relevant Update Conflict Handler](#)
- [DML Handler And a Relevant Update Conflict Handler](#)
- [Error Handler But No Relevant Update Conflict Handler](#)
- [Error Handler And a Relevant Update Conflict Handler](#)

You cannot have a DML handler and an error handler simultaneously for the same operation on the same table. Therefore, there is no scenario in which they could both be invoked.

No Relevant Handlers

If there are no relevant handlers for a row LCR, then an apply process tries to apply the change specified in the row LCR directly. If the apply process can apply the row LCR, then the change is made to the row in the table. If there is a conflict or an error during apply, then the transaction containing the row LCR is rolled back, and all of the LCRs in the transaction that satisfy the apply process rule set are moved to an exception queue.

Relevant Update Conflict Handler

Consider a case where there is a relevant update conflict handler configured, but no other relevant handlers are configured. An apply process tries to apply the change specified in a row LCR directly. If the apply process can apply the row LCR, then the change is made to the row in the table.

If there is an error during apply that is caused by a condition other than an update conflict, including a uniqueness conflict or a delete conflict, then the transaction containing the row LCR is rolled back, and all of the LCRs in the transaction that satisfy the apply process rule set are moved to an exception queue.

If there is an update conflict during apply, then the relevant update conflict handler is invoked. If the update conflict handler resolves the conflict successfully, then the apply process either applies the LCR or discards the LCR, depending on the resolution of the update conflict, and the apply process continues applying the other LCRs in the transaction that satisfy the apply process rule set. If the update conflict handler cannot resolve the conflict, then the transaction containing the row

LCR is rolled back, and all of the LCRs in the transaction that satisfy the apply process rule set are moved to an exception queue.

DML Handler But No Relevant Update Conflict Handler

Consider a case where an apply process passes a row LCR to a DML handler and there is no relevant update conflict handler configured.

The DML handler processes the row LCR. The designer of the DML handler has complete control over this processing. Some DML handlers may perform SQL operations or run the `EXECUTE` member procedure of the row LCR. If the DML handler runs the `EXECUTE` member procedure of the row LCR, then the apply process tries to apply the row LCR. This row LCR may have been modified by the DML handler.

If any SQL operation performed by the DML handler fails, or if an attempt to run the `EXECUTE` member procedure fails, then the DML handler can try to handle the exception. If the DML handler does not raise an exception, then the apply process assumes the DML handler has performed the appropriate action with the row LCR, and the apply process continues applying the other LCRs in the transaction that satisfy the apply process rule set.

If the DML handler cannot handle the exception, then the DML handler should raise an exception. In this case, the transaction containing the row LCR is rolled back, and all of the LCRs in the transaction that satisfy the apply process rule set are moved to an exception queue.

DML Handler And a Relevant Update Conflict Handler

Consider a case where an apply process passes a row LCR to a DML handler and there is a relevant update conflict handler configured.

The DML handler processes the row LCR. The designer of the DML handler has complete control over this processing. Some DML handlers may perform SQL operations or run the `EXECUTE` member procedure of the row LCR. If the DML handler runs the `EXECUTE` member procedure of the row LCR, then the apply process tries to apply the row LCR. This row LCR may have been modified by the DML handler.

If any SQL operation performed by the DML handler fails, or if an attempt to run the `EXECUTE` member procedure fails for any reason other than an update conflict, then the behavior is the same as that described in "[DML Handler But No Relevant Update Conflict Handler](#)" on page 4-18. Note that uniqueness conflicts and delete conflicts are not update conflicts.

If an attempt to run the `EXECUTE` member procedure fails because of an update conflict, then the behavior depends on the setting of the `conflict_resolution` parameter in the `EXECUTE` member procedure:

The `conflict_resolution` Parameter Is Set To `true` If the `conflict_resolution` parameter is set to `true`, then the relevant update conflict handler is invoked.

If the update conflict handler resolves the conflict successfully, and all other operations performed by the DML handler succeed, then the DML handler finishes without raising an exception and the apply process continues applying the other LCRs in the transaction that satisfy the apply process rule set.

If the update conflict handler cannot resolve the conflict, then the DML handler can try to handle the exception. If the DML handler does not raise an exception, then the apply process assumes the DML handler has performed the appropriate action with the row LCR, and the apply process continues applying the other LCRs in the transaction that satisfy the apply process rule set. If the DML handler cannot handle the exception, then the DML handler should raise an exception. In this case, the transaction containing the row LCR is rolled back, and all of the LCRs in the transaction that satisfy the apply process rule set are moved to an exception queue.

The `conflict_resolution` Parameter Is Set To `false` If the `conflict_resolution` parameter is set to `false`, then the relevant update conflict handler is not invoked. In this case, the behavior is the same as that described in "[DML Handler But No Relevant Update Conflict Handler](#)" on page 4-18.

Error Handler But No Relevant Update Conflict Handler

Consider a case where an apply process encounters an error when it tries to apply a row LCR. This error may be caused by a conflict or by some other condition. There is an error handler for the table operation but no relevant update conflict handler configured.

The row LCR is passed to the error handler. The error handler processes the row LCR. The designer of the error handler has complete control over this processing. Some error handlers may perform SQL operations or run the `EXECUTE` member procedure of the row LCR. If the error handler runs the `EXECUTE` member procedure of the row LCR, then the apply process tries to apply the row LCR. This row LCR may have been modified by the error handler.

If any SQL operation performed by the error handler fails, or if an attempt to run the EXECUTE member procedure fails, then the error handler can try to handle the exception. If the error handler does not raise an exception, then the apply process assumes the error handler has performed the appropriate action with the row LCR, and the apply process continues applying the other LCRs in the transaction that satisfy the apply process rule set.

If the error handler cannot handle the exception, then the error handler should raise an exception. In this case, the transaction containing the row LCR is rolled back, and all of the LCRs in the transaction that satisfy the apply process rule set are moved to an exception queue.

Error Handler And a Relevant Update Conflict Handler

Consider a case where an apply process encounters an error when it tries to apply a row LCR. There is an error handler for the table operation, and there is a relevant update conflict handler configured.

The handler that is invoked to handle the error depends on the type of error it is:

- If the error is caused by a condition other than an update conflict, including a uniqueness conflict or a delete conflict, then the error handler is invoked, and the behavior is the same as that described in "[Error Handler But No Relevant Update Conflict Handler](#)" on page 4-19.
- If the error is caused by an update conflict, then the update conflict handler is invoked. If the update conflict handler resolves the conflict successfully, then the apply process continues applying the other LCRs in the transaction that satisfy the apply process rule set. In this case, the error handler is not invoked.

If the update conflict handler cannot resolve the conflict, then the error handler is invoked. If the error handler does not raise an exception, then the apply process assumes the error handler has performed the appropriate action with the row LCR, and the apply process continues applying the other LCRs in the transaction that satisfy the apply process rule set. If the error handler cannot process the LCR, then the error handler should raise an exception. In this case, the transaction containing the row LCR is rolled back, and all of the LCRs in the transaction that satisfy the apply process rule set are moved to an exception queue.

See Also:

- *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about the EXECUTE member procedure for row LCRs
- ["Managing a DML Handler"](#) on page 14-14
- ["Managing an Error Handler"](#) on page 14-21
- ["Managing Streams Conflict Resolution"](#) on page 14-29

Considerations for Applying DDL Changes

The following sections discuss considerations for applying DDL changes to tables:

- [Types of DDL Changes Ignored by an Apply Process](#)
- [Database Structures in a Streams Environment](#)
- [Current Schema User Must Exist at Destination Database](#)
- [System-Generated Names](#)
- [CREATE TABLE AS SELECT Statements](#)

See Also: ["Types of DDL Changes Ignored by a Capture Process"](#) on page 2-8

Types of DDL Changes Ignored by an Apply Process

The following types of DDL changes are not supported by an apply process. These types of DDL changes are not applied:

- ALTER MATERIALIZED VIEW
- ALTER MATERIALIZED VIEW LOG
- CREATE DATABASE LINK
- CREATE SCHEMA AUTHORIZATION
- CREATE MATERIALIZED VIEW
- CREATE MATERIALIZED VIEW LOG
- DROP DATABASE LINK

- DROP MATERIALIZED VIEW
- DROP MATERIALIZED VIEW LOG
- RENAME

Also, the following types of CREATE TABLE and ALTER TABLE statements are ignored by an apply process:

- A CREATE TABLE AS SELECT statement on a clustered table is not supported in a Streams environment.
- A CREATE TABLE statement that creates an index-organized table.
- An ALTER TABLE statement that alters an index-organized table.

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 the following message in the apply process trace file, followed by the DDL text ignored:

```
Apply process ignored the following DDL:
```

An apply process applies all other types of DDL changes if the DDL LCRs containing the changes satisfy the rules in the apply process rule set. Also, an apply process can apply valid, user-enqueued DDL LCRs.

Note:

- An apply process applies ALTER *object_type object_name* RENAME changes, such as ALTER TABLE jobs RENAME. Therefore, if you want DDL changes that rename objects to be applied, then use ALTER *object_type object_name* RENAME statements instead of RENAME statements.
 - The name "materialized view" is synonymous with the name "snapshot". Snapshot equivalents of the statements on materialized views are ignored by an apply process.
-
-

Database Structures in a Streams Environment

For captured DDL changes to be applied properly at a destination database, either the destination database must have the same database structures as the source database, or the non-identical database structural information must not be specified in the DDL statement. Database structures include data files, tablespaces, rollback segments, and other physical and logical structures that support database objects.

For example, for captured DDL changes to tables to be applied properly at a destination database, the following conditions must be met:

- The same storage parameters must be specified in the `CREATE TABLE` statement at the source database and destination database.
- If a DDL statement refers to specific tablespaces or rollback segments, then the tablespaces or rollback segments must have the same names and compatible specifications at the source database and destination database.

However, if the tablespaces and rollback segments are not specified in the DDL statement, then the default tablespaces and rollback segments are used. In this case, the tablespaces and rollback segments can differ at the source database and destination database.

- The same partitioning specifications must be used at the source database and destination database.

Current Schema User Must Exist at Destination Database

For a DDL LCR to be applied at a destination database successfully, the user specified as the `current_schema` in the DDL LCR must exist at the destination database. The current schema is the schema that is used if no schema is specified for an object in the DDL text.

See Also:

- *Oracle9i Database Concepts* for more information about database structures
- *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about the `current_schema` attribute in DDL LCRs

System-Generated Names

If you plan to capture DDL changes at a source database and apply these DDL changes at a destination database, then avoid using system-generated names. If a DDL statement results in a system-generated name for an object, then the name of the object typically will be different at the source database and each destination database applying the DDL change from this source database. Different names for objects can result in apply errors for future DDL changes.

For example, suppose the following DDL statement is run at a source database:

```
CREATE TABLE sys_gen_name (n1 NUMBER NOT NULL);
```

This statement results in a NOT NULL constraint with a system-generated name. For example, the NOT NULL constraint may be named `sys_001500`. When this change is applied at a destination database, the system-generated name for this constraint may be `sys_c1000`.

Suppose the following DDL statement is run at the source database:

```
ALTER TABLE sys_gen_name DROP CONSTRAINT sys_001500;
```

This DDL statement succeeds at the source database, but it fails at the destination database and results in an apply error.

To avoid such an error, explicitly name all objects resulting from DDL statements. For example, to name a NOT NULL constraint explicitly, run the following DDL statement:

```
CREATE TABLE sys_gen_name (n1 NUMBER CONSTRAINT sys_gen_name_nn NOT NULL);
```

CREATE TABLE AS SELECT Statements

When applying a change resulting from a CREATE TABLE AS SELECT statement, an apply process performs two steps:

1. The CREATE TABLE AS SELECT statement is executed at the destination database, but it creates only the structure of the table. It does not insert any rows into the table. If the CREATE TABLE AS SELECT statement fails, then an apply process error results. Otherwise, the statement autocommits, and the apply process performs Step 2.
2. The apply process inserts the rows that were inserted at the source database as a result of the CREATE TABLE AS SELECT statement into the corresponding table at the destination database. It is possible that a capture process, a propagation, or an apply process will discard all of the row LCRs with these inserts based on their rule sets. In this case, the table remains empty at the destination database.

Note: A CREATE TABLE AS SELECT statement on a clustered table is not supported in a Streams environment.

Trigger Firing Property

You can control a DML or DDL trigger's firing property using the `SET_TRIGGER_FIRING_PROPERTY` procedure in the `DBMS_DDL` package. This procedure lets you specify whether a trigger's firing property is set to fire once. If a trigger's firing property is set to fire once, then it does not fire in the following cases:

- When a relevant change is made by an apply process
- When a relevant change results from the execution of one or more apply errors using the `EXECUTE_ERROR` or `EXECUTE_ALL_ERRORS` procedure in the `DBMS_APPLY_ADM` package

If a trigger is not set to fire once, then it fires in both of these cases.

By default, DML and DDL triggers are set to fire once. You can check a trigger's firing property by using the `IS_TRIGGER_FIRE_ONCE` function in the `DBMS_DDL` package.

For example, in the `hr` schema, the `update_job_history` trigger adds a row to the `job_history` table when data is updated in the `job_id` or `department_id` column in the `employees` table. Suppose, in a Streams environment, the following configuration exists:

- A capture process captures changes to both of these tables at the `dbs1.net` database.
- A propagation propagates these changes to the `dbs2.net` database.
- An apply process applies these changes at the `dbs2.net` database.
- The `update_job_history` trigger exists in the `hr` schema in both databases.

If the `update_job_history` trigger is not set to fire once at `dbs2.net` in this scenario, then these actions result:

1. The `job_id` column is updated for an employee in the `employees` table at `dbs1.net`.
2. The `update_job_history` trigger fires at `dbs1.net` and adds a row to the `job_history` table that records the change.
3. The capture process at `dbs1.net` captures the changes to both the `employees` table and the `job_history` table.
4. A propagation propagates these changes to the `dbs2.net` database.

5. An apply process at the `db2.net` database applies both changes.
6. The `update_job_history` trigger fires at `db2.net` when the apply process updates the `employees` table.

In this case, the change to the `employees` table is recorded twice at the `db2.net` database: when the apply process applies the change to the `job_history` table and when the `update_job_history` trigger fires to record the change made to the `employees` table by the apply process.

As you can see, the database administrator may not want the `update_job_history` trigger to fire at the `db2.net` database when a change is made by the apply process. Similarly, a database administrator may not want a trigger to fire because of the execution of an apply error transaction. If the `update_job_history` trigger's firing property is set to fire once, then it does not fire at `db2.net` when the apply process applies a change to the `employees` table, and it does not fire when an executed error transaction updates the `employees` table.

Also, if you use the `ON SCHEMA` clause to create a schema trigger, then the schema trigger fires only if the schema performs a relevant change. Therefore, when an apply process is applying changes, a schema trigger that is set to fire always fires only if the apply user is the same as the schema specified in the schema trigger. If the schema trigger is set to fire once, then it never fires when an apply process applies changes, regardless of whether the apply user is the same as the schema specified in the schema trigger.

For example, if you specify a schema trigger that always fires on the `hr` schema at a source database and destination database, but the apply user at a destination database is `stradmin`, then the trigger fires when the `hr` user performs a relevant change on the source database, but the trigger does not fire when this change is applied at the destination database. However, if you specify a schema trigger that always fires on the `stradmin` schema at the destination database, then this trigger fires whenever a relevant change is made by the apply process, regardless of any trigger specifications at the source database.

Note: Only DML and DDL triggers can be set to fire once. All other types of triggers always fire.

See Also: *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about setting a trigger's firing property with the `SET_TRIGGER_FIRING_PROPERTY` procedure

Instantiation SCN and Ignore SCN

In a Streams environment that shares information between multiple databases, a source database is the database where changes are generated in the redo log. Suppose an environment has the following characteristics:

- A capture process will capture changes to tables at the source database.
- The changes to the tables will be propagated to a destination database and applied there.
- The destination database already contains some or all of the tables for which changes will be captured, propagated, and applied.

In such an environment, the tables that already exist at the destination database are not instantiated. That is, because these tables already exist at the destination database, they are not created at the destination by exporting them at the source database and then importing them at the destination database. Instead, the apply process at the destination database must be instructed explicitly to apply changes that committed after a specific system change number (SCN) for each source database table. The **instantiation SCN** for the tables specifies this SCN.

The instantiation SCN for a database object controls which LCRs that contain changes to the database object are ignored by an apply process and which LCRs are applied by an apply process. If the commit SCN of an LCR for a database object from a source database is less than or equal to the instantiation SCN for that database object at a destination database, then the apply process at the destination database discards the LCR. Otherwise, the apply process applies the LCR.

Also, if there are multiple source databases for a shared database object at a destination database, then an instantiation SCN must be set for each source database, and the instantiation SCN may be different for each source database. You can set an instantiation SCN using export/import or using a procedure in the `DBMS_APPLY_ADM` package.

Streams also records the **ignore SCN** for each database object. The ignore SCN is the SCN below which the instantiation SCN cannot be set. This value corresponds to the SCN value at the source database at the time when the object was prepared for instantiation.

You can view the instantiation SCN and ignore SCN for database objects by querying the `DBA_APPLY_INSTANTIATED_OBJECTS` data dictionary view.

See Also:

- ["Setting Instantiation SCNs at a Destination Database"](#) on page 14-35
- ["Preparing Database Objects for Instantiation at a Source Database"](#) on page 12-11

The Oldest SCN for an Apply Process

If an apply process is running, then the **oldest SCN** is the earliest SCN of the transactions currently being dequeued and applied. For a stopped apply process, the oldest SCN is the earliest SCN of the transactions that were being applied when the apply process was stopped.

The following are two common scenarios in which the oldest SCN is important:

- You must recover the database in which the apply process is running to a certain point in time.
- You stop using an existing capture process that captures changes for the apply process and use a different capture process to capture changes for the apply process.

In both cases, you should determine the oldest SCN for the apply process by querying the `DBA_APPLY_PROGRESS` data dictionary view; the `OLDEST_MESSAGE_NUMBER` column in this view contains the oldest SCN. Then, set the start SCN for the capture process that is capturing changes for the apply process to the same value as the oldest SCN value. If the capture process is capturing changes for other apply processes, then these other apply processes may receive duplicate LCR events when you reset the start SCN for the capture process. In this case, the other apply processes automatically discard the duplicate LCR events.

See Also: ["The Start SCN, Captured SCN, and Applied SCN for a Capture Process"](#) on page 2-15

Low-Watermark and High-Watermark for an Apply Process

The **low-watermark** for an apply process is the system change number (SCN) up to which all events have been applied. That is, events that were committed at an SCN less than or equal to the low-watermark number have definitely been applied, but some events that were committed with a higher SCN also may have been applied. The low-watermark is also sometimes called the **applied SCN**.

The **high-watermark** for an apply process is the SCN beyond which no events have been applied. That is, no events that were committed with an SCN greater than the high-watermark have been applied.

You can view the low-watermark and high-watermark for one or more apply processes by querying the `V$STREAMS_APPLY_COORDINATOR` and `ALL_APPLY_PROGRESS` data dictionary views.

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 environment. If you use an apply process to apply captured LCRs in a Real Application Clusters environment, then any call to the `START_APPLY` procedure in the `DBMS_APPLY_ADM` package must be run on the owner instance of the queue that is used by the apply process.

Calls to other procedures and functions that operate on an apply process can be performed from any instance. Also, an apply process that applies user-enqueued events can start in any 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. If this happens, then, to restart the apply process, connect to the owner instance for the queue and run the `START_APPLY` procedure. The `DBA_QUEUE_TABLES` data dictionary view contains information about the owner instance for a queue table. The apply process maintains a persistent start/stop state in a Real Application Clusters environment only if the owner instance for its queue does not change before the database instance owning the queue is restarted.

Also, in a Real Application Clusters environment, an apply coordinator process, its corresponding apply reader server, and all of its apply server processes run on a single instance.

See Also:

- ["Streams Queues and Oracle Real Application Clusters"](#) on page 3-18
- ["Streams Capture Processes and Oracle Real Application Clusters"](#) on page 2-17
- *Oracle9i Database Reference* for more information about the DBA_QUEUE_TABLES data dictionary view
- ["The Persistent State of an Apply Process"](#) on page 4-36

Apply Process Architecture

You can create, alter, start, stop, and drop an apply process, and you can define apply rules that control which events an apply process dequeues from the queue. The user who creates an apply process is, by default, the user who applies changes. This user must have the necessary privileges to apply changes.

See Also: ["Configuring a Streams Administrator"](#) on page 11-2 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](#)
- [The Persistent State of an Apply Process](#)
- [Exception Queues](#)

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, which assigns them to idle apply servers.
- A **coordinator process** that gets transactions from the reader and passes them to apply servers. The coordinator process name is `apnn`, where `nn` is a coordinator process number. Valid coordinator process names include `ap01` through `ap99`.
- 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, it then tries to resolve the error with a user-specified 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 an exception queue.

When an apply server commits a completed transaction, this transaction has been applied. When an apply server places a transaction in an exception 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 and waits for instructions. The coordinator 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. After apply server A has applied the insert transaction, the coordinator process instructs apply server B to apply the update transaction.

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 are added to the rule set automatically. The `DBMS_STREAMS_ADM` package was designed for use in replication environments. Alternatively, using the `DBMS_APPLY_ADM` package to create an apply process is more flexible, and you create a rule set 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 and can apply only captured events. To create an apply process that applies events at a remote database or an apply process that applies user-enqueued events, 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 the 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: *Oracle9i Supplied 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_CAPTURE_ADM.CREATE_APPLY`

Streams Data Dictionary for an Apply Process

When a capture process is created, a duplicate data dictionary called the Streams data dictionary is populated automatically. 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 when a capture process evaluates rules and creates LCRs. 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 user 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: ["Data Dictionary Duplication During Capture Process Creation"](#) on page 2-22

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 14-11
- This section does not discuss all of the available apply process parameters. See the `DBMS_APPLY_ADM.SET_PARAMETER` procedure in the *Oracle9i Supplied 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 one 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 4-31 for more information about apply servers and the reader server
- *Oracle9i 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 from the commit order at the source database. Dependent transactions are always applied at the destination database in the same order as they were committed at the source database.

You control whether the apply servers can apply nondependent transactions in a different order at the destination database than the source 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 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 contain commit changes in a different order from 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 (`apnn`) remains the same.

Stop or Continue on Error

Using the `disable_on_error` apply process parameter, you can either 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: ["Exception Queues"](#) on page 4-37

The Persistent State of an Apply Process

An apply process maintains a persistent state. That is, an apply process maintains its current state when the database is shut down and restarted. For example, if an apply process is running when the database is shut down, then the apply process automatically starts when the database is restarted, but, if an apply process is stopped when a database is shut down, then the apply process remains stopped when the database is restarted.

Exception Queues

An exception queue is associated with each queue table. When you create a Streams queue using the `SET_UP_QUEUE` procedure in the `DBMS_STREAMS_ADM` package, an exception queue is created automatically, if one does not already exist for the queue table used by the Streams queue. To view information about Streams apply errors in all of the exception queues in a database, query the `DBA_APPLY_ERROR` data dictionary view.

An exception queue stores information about transactions that could not be applied successfully by an apply process running in the database. A transaction may include many events, and when an unhandled error occurs during apply, the apply process automatically copies all of the events in the transaction that satisfy the apply process rule set to an exception queue. The last error moved to an exception queue is on top of the error stack.

An exception queue contains information only about errors encountered at the local destination database. It does not contain information about errors for apply processes running in other databases in a Streams environment.

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 exception queue using the `EXECUTE_ERROR` and `EXECUTE_ALL_ERRORS` procedures or delete the transaction from the exception queue using the `DELETE_ERROR` and `DELETE_ALL_ERRORS` procedures. Both of these procedures are in the `DBMS_APPLY_ADM` package.

When you reexecute a transaction in an exception queue, you can specify that the transaction be executed either by the user who originally placed the error in the exception 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 an exception queue.

A reexecuted transaction uses any relevant apply handlers and conflict resolution handlers. If, to resolve the error, the LCR inside an exception queue needs to be modified before it is executed, then you can configure a DML handler to process the LCR that caused the error in the exception queue. In this case, the DML handler may modify the LCR in some way to avoid a repeat of the same error. The LCR is passed to the DML handler when you reexecute the error containing the LCR using the `EXECUTE_ERROR` or `EXECUTE_ALL_ERRORS` procedure in the `DBMS_APPLY_ADM` package.

See Also:

- ["Managing Apply Errors"](#) on page 14-33
- ["Checking for Apply Errors"](#) on page 17-36
- ["Displaying Detailed Information About Apply Errors"](#) on page 17-37
- ["Managing a DML Handler"](#) on page 14-14
- *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information on the `DBMS_APPLY_ADM` package
- *Oracle9i Database Reference* for more information about the `DBA_APPLY_ERROR` data dictionary view

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 15, "Managing Rules and Rule-Based Transformations"](#)
- [Chapter 24, "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 operators 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 condition consists of two expressions (`department_id` and `30`) and an operator (`=`):

```
department_id = 30
```

This logical condition evaluates to `TRUE` for a given row when the `department_id` column is `30`. Here, the value is data in the `department_id` column of a table.

A single rule condition may include more than one condition combined with the `AND`, `OR`, and `NOT` conditional operators to form compound conditions. For example, consider the following compound condition:

```
department_id = 30 OR job_title = 'Programmer'
```

This rule condition contains two conditions joined by the `OR` conditional operator. If either condition evaluates to `TRUE`, then the rule condition evaluates to `TRUE`. If the conditional operator were `AND` instead of `OR`, then both conditions would have to 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 `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 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
```

See Also: *Oracle9i 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 operator constant*
- *constant operator 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, an *operator* is one of the following:

- <=
- <
- =
- >
- >=

Use of other operators 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`
- `:v1 > 'aaa'`
- `:v2.a1 < 10.01`
- `:v3.m() = 10`

Rules with simple rule conditions are called simple rules. You can combine two or more simple rule conditions with the conditional operators `AND` and `OR` for a rule, and the rule remains simple. However, using the `NOT` conditional operator in a rule's condition causes the rule to be non-simple. For example, rules with the following conditions are simple rules:

- `tab1.col = 5 AND :v1 > 'aaa'`
- `tab1.col = 5 OR :v1 > 'aaa'`

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: *Oracle9i SQL Reference* for more information about conditions, expressions, and operators

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 information 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 at evaluation time. The value of an implicit variable is obtained by calling the variable value evaluation function. You define this function when you specify the `variable_types` list during the creation of an evaluation context using the `DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT` procedure. If the value for an implicit variable is specified during evaluation, then the specified value overrides the value returned by the variable value evaluation 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 evaluation 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. As you can see, 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:

- *Oracle9i Supplied 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
- *Oracle9i 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. Then, the rules engine invokes the evaluation function during the evaluation of any rule set that uses the evaluation context. The function must have each parameter in the `DBMS_RULE.EVALUATE` procedure, 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 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.

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: *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about the evaluation function specified in the `DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT` 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`. The rules engine does not interpret the action context. Instead, it returns the action context information when a rule evaluates to `TRUE`. Then, 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 the rule conditions and the action contexts for three departments:

Rule Name	Rule Condition	Action Context Name-Value Pair
<code>rule_dep_10</code>	<code>department_id = 10</code>	<code>course_number, 1057</code>
<code>rule_dep_20</code>	<code>department_id = 20</code>	<code>course_number, 1215</code>
<code>rule_dep_30</code>	<code>department_id = 30</code>	<code>NULL</code>

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 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
<code>course_number</code>	1057
<code>dist_list</code>	<code>admin_list</code>

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.

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.

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.

See Also:

- ["Row Subsetting Using Streams Rules"](#) on page 4-12
- *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about the `RE$NV_LIST` type
- ["Rule-Based Transformations"](#) on page 6-25 for examples that add and modify name-value pairs

Rule Set Evaluation

The rules engine evaluates rule sets based on events. 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. 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 evaluation 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 evaluation function (for implicit variables), and a variable method function.

The client can also send other information about the event and about how to evaluate the event 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.

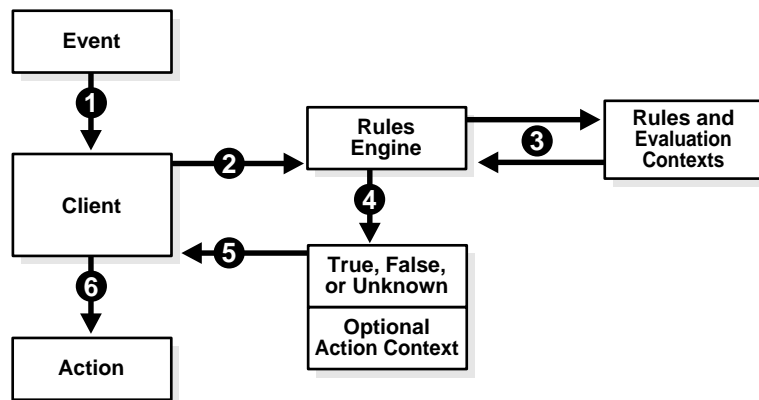
The rules engine uses the rules in the specified rule set to evaluate the event. Then, the rules engine returns the results to the client. The rules engine returns rules using the two `OUT` parameters in the `EVALUATE` procedure: `true_rules` and `maybe_rules`. That is, the `true_rules` parameter returns rules that evaluate to `TRUE`, and, optionally, the `maybe_rules` parameter returns rules 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 sends the event to the rules engine by running the `DBMS_RULE.EVALUATE` procedure.
3. The rules engine evaluates the event based on rules in the rule set and 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. 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:

- *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about the `DBMS_RULE.EVALUATE` procedure
- ["Undefined Variables in Rule Conditions Evaluate to NULL for Streams Clients"](#) on page 6-22 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 5-3

Database Objects and Privileges Related to Rules

You can create the following types of database objects directly using the `DBMS_RULE_ADM` package:

- Evaluation contexts
- Rules
- Rule sets

You can create rules and rule 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
- *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about the system and object privileges for these database objects
- *Oracle9i Database Concepts* and *Oracle9i Database Administrator's 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.

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, a user 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.

How Rules Are Used In Streams

This chapter explains how rules are used in Streams.

This chapter contains these topics:

- [Overview of How Rules Are Used In Streams](#)
- [System-Created Rules](#)
- [Streams Evaluation Context](#)
- [Streams and Event Contexts](#)
- [Streams and Action Contexts](#)
- [User-Created Rules, Rule Sets, and Evaluation Contexts](#)
- [Rule-Based Transformations](#)

See Also:

- [Chapter 5, "Rules"](#) for more information about rules
- [Chapter 15, "Managing Rules and Rule-Based Transformations"](#)

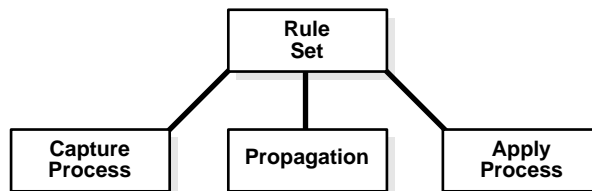
Overview of How Rules Are Used In Streams

In Streams, each of the following mechanisms is a client of a rules engine, when the mechanism is associated with a rule set:

- A capture process
- A propagation
- An apply process

Each of these mechanisms can be associated with at most one rule set. However, a single rule set can be used by multiple capture processes, propagations, and apply processes within the same database. [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*



Specifically, you use rule sets in Streams to do the following:

- Specify the changes a capture process captures from the redo log. That is, if a change found in the redo log causes any rule in the rule set associated with a capture process to evaluate to `TRUE`, then the change is captured by the capture process.
- Specify the events a propagation propagates from one queue to another. That is, if an event in a queue causes any rule in the rule set associated with a propagation to evaluate to `TRUE`, then the event is propagated by the propagation.
- Specify the events an apply process retrieves from a queue. That is, if an event in a queue causes any rule in the rule set associated with an apply process to evaluate to `TRUE`, then the event is retrieved and processed by the apply process.

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 associated with a mechanism, then the mechanism performs the task if either rule evaluates to `TRUE`. For example, if a rule set associated with a capture process contains one rule that instructs the capture process to capture DML changes to the `hr.employees` table, but another rule in the rule set instructs the capture process not to capture DML changes to the `hr.employees` table, then the capture process captures DML changes to the `hr.employees` table.

System-Created Rules

Streams performs three tasks based on rules:

- Capturing changes with a capture process
- Propagating changes with a propagation
- Applying changes with an apply process

A system-created rule specifies one of the following levels of granularity for a task: 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.

[Table 6–1](#) shows what each level of rule means for each Streams task.

Table 6–1 Types of Tasks and Rule Levels

Task	Table Rule	Schema Rule	Global Rule
Capture	Capture the changes in the redo log for the specified table, convert them into logical change records (LCRs), and enqueue them.	Capture the changes in the redo log for the database objects in the specified schema, convert them into LCRs, and enqueue them.	Capture the changes to all of the database objects in the database, convert them into LCRs, and enqueue them.
Propagate	Propagate the LCRs relating to the specified table in the source queue to the destination queue.	Propagate the LCRs related to the database objects in the specified schema in the source queue to the destination queue.	Propagate all of the changes in the source queue to the destination queue.
Apply	Apply all or a subset of the LCRs in the queue relating to the specified table.	Apply the LCRs in the queue relating to the database objects in the specified schema.	Apply all of the LCRs in the queue.

You can use procedures in the `DBMS_STREAMS_ADM` package to create rules at each of these levels. [Table 6–2](#) lists the types of system-created rule conditions specified in the rules created by the `DBMS_STREAMS_ADM` package.

Table 6–2 System-Created Rule Conditions Created by `DBMS_STREAMS_ADM` Package (Page 1 of 2)

Rule Condition Evaluates to TRUE for	Streams Mechanism	Create Using Procedure
All DML changes to a particular table	Capture	<code>ADD_TABLE_RULES</code>
All DDL changes to a particular table	Capture	<code>ADD_TABLE_RULES</code>
All DML changes to all of the tables in a particular schema	Capture	<code>ADD_SCHEMA_RULES</code>
All DDL changes to all of the database objects in a particular schema	Capture	<code>ADD_SCHEMA_RULES</code>
All DML changes to all of the tables in a particular database	Capture	<code>ADD_GLOBAL_RULES</code>
All DDL changes to all of the database objects in a particular database	Capture	<code>ADD_GLOBAL_RULES</code>
All LCRs containing DML changes to a particular table	Propagation	<code>ADD_TABLE_PROPAGATION_RULES</code>
All LCRs containing DDL changes to a particular table	Propagation	<code>ADD_TABLE_PROPAGATION_RULES</code>
All LCRs containing DML changes to the tables in a particular schema	Propagation	<code>ADD_SCHEMA_PROPAGATION_RULES</code>
All LCRs containing DDL changes to the database objects in a particular schema	Propagation	<code>ADD_SCHEMA_PROPAGATION_RULES</code>
All LCRs containing DML changes in a particular queue	Propagation	<code>ADD_GLOBAL_PROPAGATION_RULES</code>
All LCRs containing DDL changes in a particular queue	Propagation	<code>ADD_GLOBAL_PROPAGATION_RULES</code>
All LCRs containing DML changes to a subset of rows in a particular table	Apply	<code>ADD_SUBSET_RULES</code>
All LCRs containing DML changes to a particular table	Apply	<code>ADD_TABLE_RULES</code>
All LCRs containing DDL changes to a particular table	Apply	<code>ADD_TABLE_RULES</code>
All LCRs containing DML changes to the tables in a particular schema	Apply	<code>ADD_SCHEMA_RULES</code>

Table 6–2 System-Created Rule Conditions Created by DBMS_STREAMS_ADM Package (Page 2 of 2)

Rule Condition Evaluates to TRUE for	Streams Mechanism	Create Using Procedure
All LCRs containing DDL changes to the database objects in a particular schema	Apply	ADD_SCHEMA_RULES
All LCRs containing DML changes in a particular queue	Apply	ADD_GLOBAL_RULES
All LCRs containing DDL changes in a particular queue	Apply	ADD_GLOBAL_RULES

Each procedure listed in [Table 6–2](#) does the following:

- Creates a capture process, propagation, or apply process if it does not already exist.
- Creates a rule set for the specified capture process, propagation, or apply process, if a rule set does not already exist for it.
- Create zero or more rules and adds the rules to the rule set for the specified capture process, propagation, or apply process.

All of the rule sets and rules created by these procedures use the `SYS.STREAMS$_EVALUATION_CONTEXT` evaluation context, which is an Oracle-supplied evaluation context for Streams environments.

Except for `ADD_SUBSET_RULES`, these procedures create either zero, one, or two rules. If you want to perform the Streams task for only DML changes or for only DDL changes, then only one rule is created. If, however, you want to perform the Streams task for both DML 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` procedure always creates three rules for three different types of DML operations on a table: `INSERT`, `UPDATE`, and `DELETE`. The `ADD_SUBSET_RULES` procedure does not create rules for DDL changes to a table. You can use the `ADD_TABLE_RULES` procedure to create a DDL rule for 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 table, schema, and global rules in more detail.

Note: To create rules with more complex rule conditions, such as rules that use the `NOT` or `OR` conditional operators, use the `DBMS_RULE_ADM` package.

See Also:

- *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about the `DBMS_STREAMS_ADM` package and the `DBMS_RULE_ADM` package
- ["Streams Evaluation Context"](#) on page 6-15
- ["Logical Change Records \(LCRs\)"](#) on page 2-2
- ["Complex Rule Conditions"](#) on page 6-20

Table and Subset Rules

When you use a rule to specify a Streams task that is relevant only for an individual table, you are specifying a table-level rule. You can specify a table-level rule for DML changes, a table-level rule for DDL changes, or two rules for both types of changes for a specific table.

A subset rule is a special type of table-level rule for DML changes that you can create with the `ADD_SUBSET_RULES` procedure. You can use the `ADD_SUBSET_RULES` procedure to specify that an apply process only applies a subset of the row logical change records (LCRs) relating to a particular table based on a condition similar to a `WHERE` clause in a `SELECT` statement. So, the `ADD_SUBSET_RULES` procedure can instruct an apply process to maintain only certain rows in a table.

Note: Creating subset rules for tables that have one or more LOB columns is not supported.

See Also: ["Row Subsetting Using Streams Rules"](#) on page 4-12 for more information about subset rules

Table-Level Rules Example

Suppose you use the procedures in the `DBMS_STREAMS_ADM` package to instruct a Streams apply process to behave in the following ways:

- [Apply All DML Changes to the hr.locations Table](#)
- [Apply All DDL Changes to the hr.countries Table](#)
- [Apply a Subset of DML changes to the hr.regions Table](#)

Apply All DML Changes to the hr.locations Table These changes originated at the `dbssl.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     => 'dbssl.net');
END;
/
```

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' )
```

Here, every condition that begins with `:dml` is a variable. The value is determined by a call to the specified member function for the row LCR being evaluated. So,

`:dml.get_object_owner()` in the previous example is a call to the `GET_OBJECT_OWNER` member function for the row LCR being evaluated.

Also, the following condition is included by default in all DML rules created by the procedures in the `DBMS_STREAMS_ADM` package:

```
:dml.is_null_tag() = 'Y'
```

In DDL rules, the condition is the following:

```
:ddl.is_null_tag() = 'Y'
```

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. You can omit this 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.

See Also:

- [Chapter 5, "Rules"](#) for more information about variables in conditions
- [Chapter 8, "Streams Tags"](#) for more information about tags

Apply All DDL Changes to the `hr.countries` Table These changes originated at the `dbst1.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     => 'dbst1.net');
END;
/
```

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' )
```

Here, every condition that begins with `:ddl` is a variable. The value is determined by a call to the specified member function for the DDL LCR being evaluated. So, `:ddl.get_object_owner()` in the previous example is a call to the `GET_OBJECT_OWNER` member function for the DDL LCR being evaluated.

Apply a Subset of DML changes to the `hr.regions` Table This example instructs a Streams apply process to apply a subset of DML changes to the `hr.regions` table where the `region_id` is 2. These changes originated at the `dbsl.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      => 'dbsl.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, LCRs are evaluated in the following ways:

- For an insert, if the new value in the LCR for `region_id` is 2, then the insert is applied.
- For an insert, if the new value in the 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 LCR for `region_id` is not 2 or is `NULL` and the new value in the LCR for `region_id` is 2, then the update is converted into an insert and applied.

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, LCRs are evaluated in the following ways:

- For an update, if both the old value and the new value in the 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 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, LCRs are evaluated in the following ways:

- For a delete, if the old value in the LCR for `region_id` is 2, then the delete is applied.
- For a delete, if the old value in the 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 LCR for `region_id` is 2 and the new value in the LCR for `region_id` is not 2 or is `NULL`, then the update is converted into a delete and applied.

Summary of Rules In this example, the following table and subset rules were defined:

- A table rule that evaluates to `TRUE` if a DML operation is performed on the `hr.locations` table.
- A table rule that evaluates to `TRUE` if a DDL operation is performed on the `hr.countries` table.
- A subset rule that evaluates to `TRUE` if an `INSERT` operation inserts a row with a `region_id` equal to 2 or an update operation changes the `region_id` for a row from a value that does not equal 2 or is `NULL` to a value of 2.
- A subset rule that evaluates to `TRUE` if an `UPDATE` operation updates a row and the `region_id` is equal to 2 both before and after the update.
- A subset rule that evaluates to `TRUE` if a `DELETE` operation deletes a row with a `region_id` equal to 2 or an update operation changes the `region_id` for a row from a value that equals 2 to a value that does not equal 2 or is `NULL`.

Given these rules, the following list provides examples of changes applied by an apply process:

- A row is inserted to the `hr.locations` table.
- Five rows are deleted from the `hr.locations` table.
- A column is added to the `hr.countries` table.
- A row is updated in the `hr.regions` table where the `region_id` is 2 and the new value of `region_id` is 1. This update is transformed into a delete.

The apply process dequeues these changes from its associated queue and applies them to the database objects at the destination database.

Given the same 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 the `hr.employees` table does not satisfy any of the rules.
- A row is updated into 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.
- 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.

Schema Rules

When you use a rule to specify a Streams task that is relevant to a schema, you are specifying a schema-level rule, and the Streams task is performed when there is a change to any of the database objects currently in the schema and any database objects added to the schema in the future. You can specify a schema-level rule for DML changes, a schema-level rule for DDL changes, or two rules for both types of changes for the objects in the schema.

Schema-Level Rule Example

Suppose you use the `ADD_SCHEMA_PROPAGATION_RULES` procedure in the `DBMS_STREAMS_ADM` package to instruct a Streams propagation to propagate LCRs that contain DML and DDL changes 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 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');
END;
/
```

The `ADD_SCHEMA_PROPAGATION_RULES` procedure creates two rules: one for row LCRs (which contain 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' and :ddl.is_null_tag() = 'Y'
and :ddl.get_source_database_name() = 'DBS1.NET')
```

Given these rules, the following list provides examples of changes propagated by the propagation:

- A row is inserted to 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-level rule, and the `oe.inventories` table was not specified in a table-level rule.

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-level rule. You can specify a global rule for DML changes, a global rule for DDL changes, or two rules for both types of changes.

A single global rule for the capture process means that the capture process captures either all DML changes or all DDL changes to the source database. A single global rule 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 global rule 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.

Global-Level 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 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);
END;
/
```

`NULL` can be specified for the `source_database` parameter because capture rules are being created. When creating apply rules using `ADD_GLOBAL_RULES`, specify a source database name.

The `ADD_GLOBAL_RULES` procedure creates two rules: one for row LCRs (which contain DML changes) and one for DDL LCRs.

Here is the rule condition used by the row LCR rule:

```
(( :dml.get_source_database_name()>= ' ' OR :dml.get_source_database_name()<= ' ' )  
and :dml.is_null_tag() = 'Y' )
```

Here is the rule condition used by the DDL LCR rule:

```
(( :ddl.get_source_database_name()>= ' ' OR :ddl.get_source_database_name()<= ' ' )  
and :ddl.is_null_tag() = 'Y' )
```

Given these rules, the capture process captures all supported DML and DDL changes made to the database. The conditions relating the source database name in global system generated rules are present to improve rule evaluation performance.

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

See Also: [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

Streams Evaluation Context

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.

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 operator 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` procedure.

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

See Also: *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about LCRs and their member functions

Streams and Event Contexts

In Streams, capture processes 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. A propagation and an apply process always sends an event context with the following a 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 rules that refer to Streams events using implicit variables. Then, the variable value evaluation 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 evaluation function returns a value based on the event itself. You can also pass the event context to an evaluation function and a variable method function.

See Also:

- ["Rule Set Evaluation"](#) on page 5-12 for more information about event contexts
- ["Explicit and Implicit Variables"](#) on page 5-6 for more information about variable value evaluation functions
- ["Evaluation Function"](#) on page 5-8

Streams and Action Contexts

In Streams, an action context has two purposes: for internal LCR transformations in subset rules and for user-defined rule-based transformations. If an action context for a rule contains both a subset transformation and a user-defined rule-based transformation, then the subset transformation is performed before the user-defined rule-based transformation.

If you use a non-NULL action context for one or more rules in a rule set, either by specifying a subset rule or a rule-based transformation, then make sure only one rule can evaluate to `TRUE` for a particular 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 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 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:

- ["Row Subsetting Using Streams Rules"](#) on page 4-12 and ["Table and Subset Rules"](#) on page 6-6 for more information about subset rules
- ["Rule-Based Transformations"](#) on page 6-25

User-Created Rules, Rule Sets, and Evaluation Contexts

If you need to create rules that are more complex than those created by the `DBMS_STREAMS_ADM` package, then you can use the `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 specify complex rule conditions, such as those that use the `NOT` conditional operator or those that pertain only to specific operations.
- 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, or apply process.

See Also:

- ["Specifying the Rule Set for a Capture Process"](#) on page 12-5
- ["Specifying the Rule Set for a Propagation"](#) on page 13-14
- ["Specifying the Rule Set for an Apply Process"](#) on page 14-8

Complex Rule Conditions

In a Streams environment, a complex rule condition is one that cannot be created using the `DBMS_STREAMS_ADM` package. [Table 6-2](#) on page 6-4 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 more complex conditions, then use the `DBMS_RULE_ADM` package.

There are a wide range of complex conditions. The following sections contain some examples of complex rule conditions.

Note:

- 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 Conditional Operator to Exclude Objects

You can use the `NOT` conditional operator to exclude certain changes from being captured, propagated, or applied 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` conditional operator 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' 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' 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.

Rule Conditions for Specific Types of Operations

In some cases, you may want to capture, propagate, or apply 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:

```
:dml.get_command_type() != 'DELETE' AND :dml.get_object_owner() = 'HR'  
AND :dml.get_object_name() = 'DEPARTMENTS' AND :dml.is_null_tag() = 'Y'
```

Undefined Variables in Rule Conditions Evaluate to NULL for Streams Clients

During evaluation, an implicit variable in a rule condition is undefined if the variable value evaluation 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, the 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, and apply processes. In contrast, for non-Streams clients of the rules engine, an undefined variable in a rule condition may cause the rules engine to return a maybe rule 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, if the 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 5-12

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: ["Data Dictionary Duplication During Capture Process Creation"](#) on page 2-22

Custom 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 evaluation 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: *Oracle9i Database Reference* for more information about these data dictionary views

Rule-Based Transformations

In Streams, a **rule-based transformation** is any modification to an event containing an LCR that results when a rule evaluates to `TRUE`. 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.

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. Also, the LCR returned by the function must be the same LCR passed to the function. Although you can modify an LCR with a rule-based transformation, constructing a new LCR and returning it is not allowed.

Other examples of transformations on LCRs 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

In Streams, you use a rule action context to specify a rule-based 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.

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 or propagation, then the user that calls the transformation function is the user that created the capture process or propagation.
- If a transformation is specified on a rule used by an apply process, then the user that calls the transformation function is the apply user for the apply process.

When a rule evaluates to `TRUE` for an event containing an LCR 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$_`.

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 changes are being captured, propagated, or applied. The following sections describe how rule-based transformations work with a capture process, a propagation, and an apply process.

Note:

- 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 you have a large number of transformations, or transformations that are expensive, then you may want to make modifications to events 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 modification. For example, if the rule-based transformation changes the name of a table in the DDL LCR, then the table name in the DDL text should be changed 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.
-

See Also:

- ["Managing Rule-Based Transformations"](#) on page 15-11
- ["Rule Action Context"](#) on page 5-9
- *Oracle9i Application Developer's Guide - Advanced Queuing* for more information about using the `DBMS_TRANSFORM` package
- *Oracle9i Supplied PL/SQL Packages and Types Reference* for reference information on the `DBMS_TRANSFORM` package
- ["Message Propagation and SYS.AnyData Queues"](#) on page 3-16 for examples of using the `DBMS_TRANSFORM` package to perform a transformation during propagation
- ["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 rule set, then both of the following conditions must be met in order for a transformation to be performed during capture:

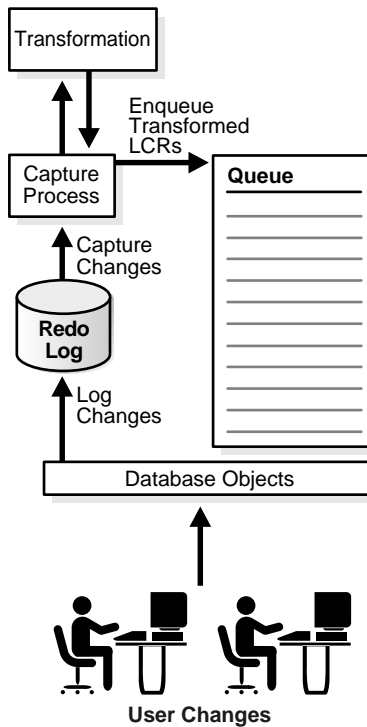
- A rule evaluates to `TRUE` for a particular change found in the redo log.
- An action context containing a name-value pair with the name `STREAMS$_TRANSFORM_FUNCTION` is returned to the capture process when the rule is evaluated.

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

Figure 6–2 shows a transformation during capture.

Figure 6–2 Transformation During Capture



For example, if an LCR event is transformed during capture, then the transformed LCR event is enqueued into the source queue. Therefore, if such a captured LCR event is propagated from the `db1.net` database to the `db2.net` and the `db3.net` databases, then the queues at `db2.net` and `db3.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:

- All sites receive the transformed LCR event.
- The transformation overhead occurs in the source database.

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

- A rule evaluates to `TRUE` for an LCR event in the source queue for the propagation. This LCR 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.

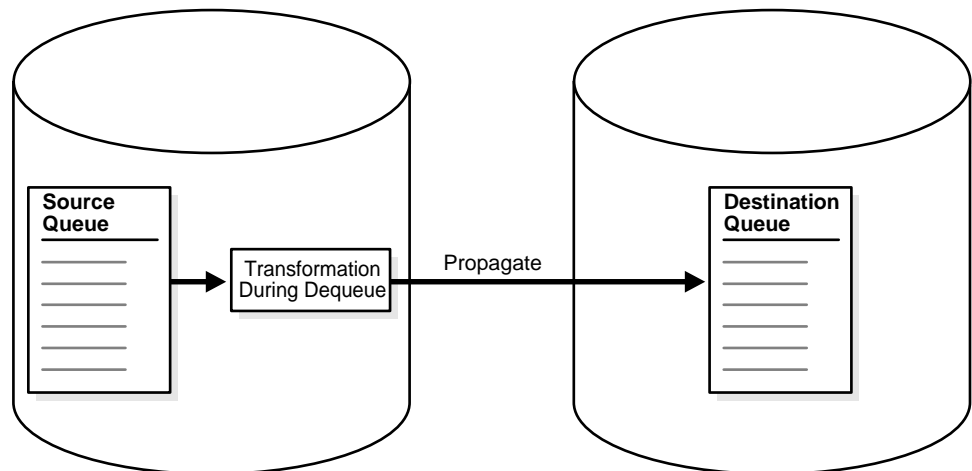
See Also: ["Captured and User-Enqueued Events"](#) on page 3-3

Given these conditions, the propagation completes the following steps:

1. Starts dequeuing the LCR event from the source queue
2. Runs the PL/SQL function in the name-value pair to transform the LCR event
3. Completes dequeuing the transformed LCR event
4. Propagates the transformed LCR event to the destination queue

Figure 6–3 shows a transformation during propagation.

Figure 6–3 Transformation During Propagation



For example, suppose you use a rule-based transformation for a propagation from the `db1.net` database to the `db2.net` database, but you do not use a rule-based transformation for a propagation from the `db1.net` database to the `db3.net` database.

In this case, an LCR event in the queue at `db1.net` can be transformed before it is propagated to `db2.net`, but the same LCR event can remain in its original form when it is propagated to `db3.net`. In this case, after propagation, the queue at `db2.net` contains the transformed LCR event, and the queue at `db3.net` contains the original LCR event.

The advantages of performing transformations during propagation are the following:

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

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

- Once an LCR event is transformed, any database to which it is propagated after the first propagation receives the transformed LCR event. For example, if `db2.net` propagates the LCR event to `db4.net`, then `db4.net` receives the transformed LCR event.
- When the first propagation in a directed network performs the transformation, the transformation overhead occurs on 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 LCR that caused the error is not dequeued, the LCR is not propagated, and the error is returned to the propagation. Before the LCR 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 rule set, then both of the following conditions must be met in order for a transformation to be performed during apply:

- A rule evaluates to `TRUE` for an LCR event in the queue associated with the apply process. This LCR 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.

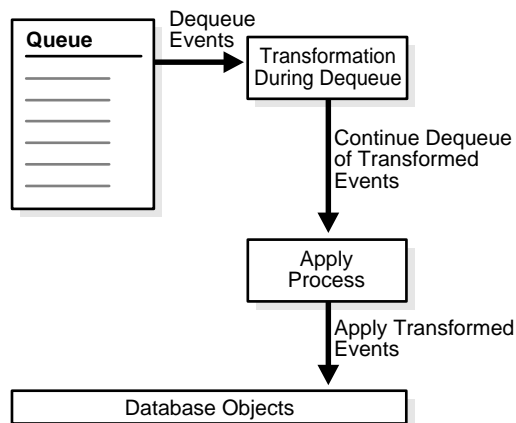
See Also: ["Captured and User-Enqueued Events"](#) on page 3-3

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

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

Figure 6–4 shows a transformation during apply.

Figure 6–4 Transformation During Apply



For example, suppose an LCR event is propagated from the `db1.net` database to the `db2.net` database in its original form. When the apply process dequeues the LCR event from a queue at `db2.net`, the LCR event is transformed.

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

- Any database to which the LCR event is propagated after the first propagation can receive the LCR event in its original form. For example, if `db2.net` propagates the LCR event to `db4.net`, then `db4.net` can receive the original LCR 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 LCR events contain private information, because all databases to which the LCR events are propagated receive the original LCR events.
- The same transformation may be done multiple times when multiple destination databases need the same transformation.

Rule-Based Transformation Errors During Apply Process Dequeue

If an error occurs when the transformation function is run during apply process dequeue, then the LCR that caused the error is not dequeued, the transaction containing the LCR 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 LCRs

If an apply error occurs for a transaction in which some of the LCRs have been transformed by a rule-based transformation, then the transformed LCRs are moved to an exception queue with all of the other LCRs in the transaction. If you use the `EXECUTE_ERROR` procedure in the `DBMS_APPLY_ADM` package to reexecute a transaction in an exception queue that contains transformed LCRs, then the transformation is not performed on the LCRs again because the apply process rule set is not evaluated again.

Multiple Rule-Based Transformations

You can transform an LCR during capture, propagation, or apply, or during any combination of capture, propagation, and apply. For example, if you want to hide sensitive data from all recipients, then you can transform an LCR during capture. If some recipients require additional custom transformations, then you can transform the previously transformed LCR during propagation or apply.

Streams Conflict Resolution

Some Streams environments must use conflict handlers to resolve possible data conflicts that can result from sharing data between multiple databases.

This chapter contains these topics:

- [About DML Conflicts in a Streams Environment](#)
- [Conflict Types in a Streams Environment](#)
- [Conflicts and Transaction Ordering in a Streams Environment](#)
- [Conflict Detection in a Streams Environment](#)
- [Conflict Avoidance in a Streams Environment](#)
- [Conflict Resolution in a Streams Environment](#)

See Also: ["Managing Streams Conflict Resolution"](#) on page 14-29

About DML Conflicts in a Streams Environment

Conflicts can occur in a Streams environment that permits concurrent data manipulation language (DML) operations on the same data at multiple databases. In a Streams environment, DML conflicts can occur only when an apply process is applying an event that contains a change resulting from a DML operation. This type of event is called a row logical change record, or row LCR. An apply process automatically detects conflicts caused by row LCRs.

For example, when two transactions originating from different databases update the same row at nearly the same time, a conflict can occur. When you configure a Streams environment, you must consider whether conflicts can occur. You can configure conflict resolution to resolve conflicts automatically, if your system design permits conflicts.

In general, you should try to design a Streams environment that avoids the possibility of conflicts. Using the conflict avoidance techniques discussed later in this chapter, most system designs can avoid conflicts in all or a large percentage of the shared data. However, many applications require that some percentage of the shared data be updatable at multiple databases at any time. If this is the case, then you must address the possibility of conflicts.

Note: An apply process does not detect DDL conflicts or conflicts resulting from user-enqueued events. Make sure your environment avoids these types of conflicts.

See Also: ["Row LCRs"](#) on page 2-3

Conflict Types in a Streams Environment

You may encounter these types of conflicts when you share data at multiple databases:

- [Update Conflicts in a Streams Environment](#)
- [Uniqueness Conflicts in a Streams Environment](#)
- [Delete Conflicts in a Streams Environment](#)
- [Foreign Key Conflicts in a Streams Environment](#)

Update Conflicts in a Streams Environment

An **update conflict** occurs when the apply process applies a row LCR containing an update to a row that conflicts with another update to the same row. Update conflicts can happen when two transactions originating from different databases update the same row at nearly the same time.

Uniqueness Conflicts in a Streams Environment

A **uniqueness conflict** occurs when the apply process applies a row LCR containing a change to a row that violates a uniqueness integrity constraint, such as a `PRIMARY KEY` or `UNIQUE` constraint. For example, consider what happens when two transactions originate from two different databases, each inserting a row into a table with the same primary key value. In this case, the transactions cause a uniqueness conflict.

Delete Conflicts in a Streams Environment

A **delete conflict** occurs when two transactions originate from different databases, with one transaction deleting a row and another transaction updating or deleting the same row. In this case, the row referenced in the row LCR does not exist to be either updated or deleted.

Foreign Key Conflicts in a Streams Environment

A **foreign key conflict** occurs when the apply process applies a row LCR containing a change to a row that violates a foreign key constraint. For example, in the `hr` schema, the `department_id` column in the `employees` table is a foreign key of the `department_id` column in the `departments` table. Consider what can happen when the following changes originate from two different databases (A and B) and are propagated to a third database (C):

- At database A, a row is inserted into the `departments` table with a `department_id` of 271. This change is propagated to database B and applied there.
- At database B, a row is inserted into the `employees` table with an `employee_id` of 206 and a `department_id` of 271.

If the change that originated at database B is applied at database C before the change that originated at database A, then a foreign key conflict results because the row for the department with a `department_id` of 271 does not yet exist in the `departments` table at database C.

Conflicts and Transaction Ordering in a Streams Environment

Ordering conflicts can occur in a Streams environment when three or more databases share data and the data is updated at two or more of these databases. For example, consider a scenario in which three databases share information in the `hr.departments` table. The database names are `mult1.net`, `mult2.net`, and `mult3.net`. Suppose a change is made to a row in the `hr.departments` table at `mult1.net` that will be propagated to both `mult2.net` and `mult3.net`. The following series of actions may occur:

1. The change is propagated to `mult2.net`.
2. An apply process at `mult2.net` applies the change from `mult1.net`.
3. A different change to the same row is made at `mult2.net`.
4. The change at `mult2.net` is propagated to `mult3.net`.
5. An apply process at `mult3.net` attempts to apply the change from `mult2.net` before another apply process at `mult3.net` applies the change from `mult1.net`.

In this case, a conflict occurs because a column value for the row at `mult3.net` does not match the corresponding old value in the row LCR propagated from `mult2.net`.

In addition to causing a data conflict, transactions that are applied out of order might experience referential integrity problems at a remote database if supporting data has not been successfully propagated to that database. Consider the scenario where a new customer calls an order department. A customer record is created and an order is placed. If the order data is applied at a remote database before the customer data, then a referential integrity error is raised because the customer that the order references does not exist at the remote database.

If an ordering conflict is encountered, then you can resolve the conflict by reexecuting the transaction in an exception queue after the required data has been propagated to the remote database and applied.

Conflict Detection in a Streams Environment

An apply process detects update, uniqueness, delete, and foreign conflicts as follows:

- An apply process detects an update conflict if there is any difference between the old values for a row in a row LCR and the current values of the same row at the destination database.
- An apply process detects a uniqueness conflict if a uniqueness constraint violation occurs when applying an LCR that contains an insert or update operation.
- An apply process detects a delete conflict if it cannot find a row when applying an LCR that contains an update or delete operation, because the primary key of the row does not exist.
- An apply process detects a foreign key conflict if a foreign key constraint violation occurs when applying an LCR.

A conflict may be detected when an apply process attempts to apply an LCR directly or when an apply process handler, such as a DML handler, runs the EXECUTE member procedure for an LCR. A conflict also may be detected when either the EXECUTE_ERROR or EXECUTE_ALL_ERRORS procedure in the DBMS_APPLY_ADM package is run.

Note: Any old LOB values in update LCRs, delete LCRs, and LCRs dealing with piecewise updates to LOB columns are not used by conflict detection.

Identifying Rows During Conflict Detection in a Streams Environment

To detect conflicts accurately, Oracle must be able to identify and match corresponding rows at different databases uniquely. By default, Oracle uses the primary key of a table to identify rows in the table uniquely. When a table does not have a primary key, you must designate a substitute key. A substitute key is a column or set of columns that Oracle can use to identify uniquely rows in the table.

See Also: ["Substitute Key Columns"](#) on page 4-11

Conflict Avoidance in a Streams Environment

This section describes ways to avoid data conflicts.

Use a Primary Database Ownership Model

You can avoid the possibility of conflicts by limiting the number of databases in the system that have simultaneous update access to the tables containing shared data. Primary ownership prevents all conflicts, because only a single database permits updates to a set of shared data. Applications can even use row and column subsetting to establish more granular ownership of data than at the table level. For example, applications might have update access to specific columns or rows in a shared table on a database-by-database basis.

Avoid Specific Types of Conflicts

If a primary database ownership model is too restrictive for your application requirements, then you can use a shared ownership data model, which means that conflicts may be possible. Even so, typically you can use some simple strategies to avoid specific types of conflicts.

Avoid Uniqueness Conflicts in a Streams Environment

You can avoid uniqueness conflicts by ensuring that each database uses unique identifiers for shared data. There are three ways to ensure unique identifiers at all databases in a Streams environment.

One way is to construct a unique identifier by executing the following select statement:

```
SELECT SYS_GUID() OID FROM DUAL;
```

This SQL operator returns a 16-byte globally unique identifier. This value is based on an algorithm that uses time, date, and the computer identifier to generate a globally unique identifier. The globally unique identifier appears in a format similar to the following:

```
A741C791252B3EA0E034080020AE3E0A
```

Another way to avoid uniqueness conflicts is to create a sequence at each of the databases that shares data and concatenate the database name (or other globally unique value) with the local sequence. This approach helps to avoid any duplicate sequence values and helps to prevent uniqueness conflicts.

Finally, you can create a customized sequence at each of the databases that shares data so that no two databases can generate the same value. You can accomplish this by using a combination of starting, incrementing, and maximum values in the `CREATE SEQUENCE` statement. For example, you might configure the following sequences:

Parameter	Database A	Database B	Database C
START WITH	1	3	5
INCREMENT BY	10	10	10
Range Example	1, 11, 21, 31, 41,...	3, 13, 23, 33, 43,...	5, 15, 25, 35, 45,...

Using a similar approach, you can define different ranges for each database by specifying a `START WITH` and `MAXVALUE` that would produce a unique range for each database.

Avoid Delete Conflicts in a Streams Environment

Always avoid delete conflicts in shared data environments. In general, applications that operate within a shared ownership data model should not delete rows using `DELETE` statements. Instead, applications should mark rows for deletion and then configure the system to purge logically deleted rows periodically.

Avoid Update Conflicts in a Streams Environment

After trying to eliminate the possibility of uniqueness and delete conflicts, you should also try to limit the number of possible update conflicts. However, in a shared ownership data model, update conflicts cannot be avoided in all cases. If you cannot avoid all update conflicts, then you must understand the types of conflicts possible and configure the system to resolve them if they occur.

Conflict Resolution in a Streams Environment

After an update conflict has been detected, a conflict handler can attempt to resolve it. Streams provides prebuilt conflict handlers to resolve update conflicts, but not uniqueness, delete, foreign key, or ordering conflicts. However, you can build your own custom conflict handler to resolve data conflicts specific to your business rules. Such a conflict handler can be part of a DML handler or an error handler.

Whether you use a prebuilt or custom conflict handlers, a conflict handler is applied as soon as a conflict is detected. If neither the specified conflict handler nor the relevant apply handler can resolve the conflict, then the conflict is logged in an exception queue. You may want to use the relevant apply handler to notify the database administrator when a conflict occurs.

When a conflict causes a transaction to be moved to an exception queue, sometimes it is possible to correct the condition that caused the conflict. In these cases, you can reexecute a transaction using the `EXECUTE_ERROR` procedure in the `DBMS_APPLY_ADM` package.

See Also:

- ["Event Processing Options"](#) on page 4-4 for more information about DML handlers and error handlers
- ["Handlers and Row LCR Processing"](#) on page 4-16 for more information about how update conflict handlers interact with DML handlers and error handlers
- ["Exception Queues"](#) on page 4-37
- *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about the `EXECUTE_ERROR` procedure in the `DBMS_APPLY_ADM` package

Prebuilt Update Conflict Handlers

This section describes the types of prebuilt update conflict handlers available to you and how column lists and resolution columns are used in prebuilt update conflict handlers. A column list is a list of columns for which the update conflict handler is called when there is an update conflict. The resolution column is the column used to identify an update conflict handler. If you use a `MAXIMUM` or `MINIMUM` prebuilt update conflict handler, then the resolution column is also the column used to resolve the conflict. The resolution column must be one of the columns in the column list for the handler.

Use the `SET_UPDATE_CONFLICT_HANDLER` procedure in the `DBMS_APPLY_ADM` package to specify one or more update conflict handlers for a particular table. There are no prebuilt conflict handlers for uniqueness, delete, or foreign key conflicts.

See Also:

- ["Managing Streams Conflict Resolution"](#) on page 14-29 for instructions on adding, modifying, and removing an update conflict handler
- *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about the `SET_UPDATE_CONFLICT_HANDLER` procedure
- ["Column Lists"](#) on page 7-12
- ["Resolution Columns"](#) on page 7-14

Types of Prebuilt Update Conflict Handlers

Oracle provides the following types of prebuilt update conflict handlers for a Streams environment: `OVERWRITE`, `DISCARD`, `MAXIMUM`, and `MINIMUM`.

The description for each type of handler later in this section refers to the following conflict scenario:

1. The following update is made at the `dbs1.net` source database:

```
UPDATE hr.employees SET salary = 4900 WHERE employee_id = 200;
COMMIT;
```

This update changes the salary for employee 200 from 4400 to 4900.

2. At nearly the same time, the following update is made at the `dbs2.net` destination database:

```
UPDATE hr.employees SET salary = 5000 WHERE employee_id = 200;
COMMIT;
```

3. A capture process captures the update at the `dbs1.net` source database and puts the resulting row LCR in a queue.

4. A propagation propagates the row LCR from the queue at `db1.net` to a queue at `db2.net`.
5. An apply process at `db2.net` attempts to apply the row LCR to the `hr.employees` table but encounters a conflict because the salary value at `db2.net` is 5000, which does not match the old value for the salary in the row LCR (4400).

The following sections describe each prebuilt conflict handler and explain how the handler resolves this conflict.

OVERWRITE When a conflict occurs, the **OVERWRITE** handler replaces the current value at the destination database with the new value in the LCR from the source database.

If the **OVERWRITE** handler is used for the `hr.employees` table at the `db2.net` destination database in the conflict example, then the new value in the row LCR overwrites the value at `db2.net`. Therefore, after the conflict is resolved, the salary for employee 200 is 4900.

DISCARD When a conflict occurs, the **DISCARD** handler ignores the values in the LCR from the source database and retains the value at the destination database.

If the **DISCARD** handler is used for the `hr.employees` table at the `db2.net` destination database in the conflict example, then the new value in the row LCR is discarded. Therefore, after the conflict is resolved, the salary for employee 200 is 5000 at `db2.net`.

MAXIMUM When a conflict occurs, the **MAXIMUM** conflict handler compares the new value in the LCR from the source database with the current value in the destination database for a designated resolution column. If the new value of the resolution column in the LCR is greater than the current value of the column at the destination database, then the apply process resolves the conflict in favor of the LCR. If the new value of the resolution column in the LCR is less than the current value of the column at the destination database, then the apply process resolves the conflict in favor of the destination database.

If the **MAXIMUM** handler is used for the `salary` column in the `hr.employees` table at the `db2.net` destination database in the conflict example, then the apply process does not apply the row LCR, because the salary in the row LCR is less than the current salary in the table. Therefore, after the conflict is resolved, the salary for employee 200 is 5000 at `db2.net`.

If you want to resolve conflicts based on the time of the transactions involved, then one way to do this is to add a column to a shared table that automatically records the transaction time with a trigger. Then, you can designate this column as a resolution column for a `MAXIMUM` conflict handler, and the transaction with the latest (or greater) time would be used automatically.

The following is an example of a trigger that records the time of a transaction for the `hr.employees` table. Assume that the `job_id`, `salary`, and `commission_pct` columns are part of the column list for the conflict resolution handler. The trigger should fire only when an `UPDATE` is performed on the columns in the column list or when an `INSERT` is performed.

```
CONNECT hr/hr

ALTER TABLE hr.employees ADD (time TIMESTAMP WITH TIME ZONE);

CREATE OR REPLACE TRIGGER hr.insert_time_employees
BEFORE
  INSERT OR UPDATE OF job_id, salary, commission_pct ON hr.employees
FOR EACH ROW
BEGIN
  -- Consider time synchronization problems. The previous update to this
  -- row may have originated from a site with a clock time ahead of the
  -- local clock time.
  IF :OLD.TIME IS NULL OR :OLD.TIME < SYSTIMESTAMP THEN
    :NEW.TIME := SYSTIMESTAMP;
  ELSE
    :NEW.TIME := :OLD.TIME + 1 / 86400;
  END IF;
END;
/
```

If you use such a trigger for conflict resolution, then make sure the trigger's firing property is fire once, which is the default. Otherwise, a new time may be marked when transactions are applied by an apply process, resulting in the loss of the actual time of the transaction.

See Also: ["Trigger Firing Property"](#) on page 4-25

MINIMUM When a conflict occurs, the `MINIMUM` conflict handler compares the new value in the LCR from the source database with the current value in the destination database for a designated resolution column. If the new value of the resolution column in the LCR is less than the current value of the column at the destination database, then the apply process resolves the conflict in favor of the LCR. If the new value of the resolution column in the LCR is greater than the current value of the column at the destination database, then the apply process resolves the conflict in favor of the destination database.

If the `MINIMUM` handler is used for the `salary` column in the `hr.employees` table at the `dbs2.net` destination database in the conflict example, then the apply process resolves the conflict in favor of the row LCR, because the salary in the row LCR is less than the current salary in the table. Therefore, after the conflict is resolved, the salary for employee 200 is 4900.

Column Lists

Each time you specify a prebuilt update conflict handler for a table, you must specify a **column list**. A column list is a list of columns for which the update conflict handler is called. If an update conflict occurs for one or more of the columns in the list when an apply process tries to apply a row LCR, then the update conflict handler is called to resolve the conflict. The update conflict handler is not called if a conflict occurs only in columns that are not in the list. The scope of conflict resolution is a single column list on a single row LCR.

You can specify more than one update conflict handler for a particular table, but the same column cannot be in more than one column list. For example, suppose you specify two prebuilt update conflict handlers on `hr.employees` table:

- The first update conflict handler has the following columns in its column list:
`salary` and `commission_pct`.
- The second update conflict handler has the following columns in its column list:
`job_id` and `department_id`.

Also, assume that no other conflict handlers exist for this table. In this case, if a conflict occurs for the `salary` column when an apply process tries to apply a row LCR, then the first update conflict handler is called to resolve the conflict. If, however, a conflict occurs for the `department_id` column, then the second update conflict handler is called to resolve the conflict. If a conflict occurs for a column that is not in a column list for any conflict handler, then no conflict handler is called, and an error results. In this example, if a conflict occurs for the `manager_id` column, then an error results. If conflicts occur in more than one column list when a row LCR is being applied, and there are no conflicts in any columns that are not in a

column list, then the appropriate update conflict handler is invoked for each column list with a conflict.

Column lists enable you to use different handlers to resolve conflicts for different types of data. For example, numeric data is often suited for a maximum or minimum conflict handler, while an overwrite or discard conflict handler might be preferred for character data.

If a conflict occurs in a column that is not in a column list, then the error handler for the specific operation on the table attempts to resolve the conflict. If the error handler cannot resolve the conflict, or if there is no such error handler, then the transaction that caused the conflict is moved to an exception queue.

Also, if a conflict occurs for a column in a column list that uses either the `OVERWRITE`, `MAXIMUM`, or `MINIMUM` prebuilt handler, and the row LCR does not contain all of the columns in this column list, then the conflict cannot be resolved because all of the values are not available. In this case, the transaction that caused the conflict is moved to an exception queue. If the column list uses the `DISCARD` prebuilt method, then the row LCR is discarded and no error results, even if the row LCR does not contain all of the columns in this column list.

A conditional supplemental log group must be specified for the columns specified in a column list if more than one column at the source database affects the column list at the destination database. Supplemental logging is specified at the source database and adds additional information to the LCR, which is needed to resolve conflicts properly. Typically, a conditional supplemental log group must be specified for the columns in a column list if there is more than one column in the column list, but not if there is only one column in the column list.

However, in some cases, a conditional supplemental log group is required even if there is only one column in a column list. That is, an apply handler or rule-based transformation may combine multiple columns from the source database into a single column in the column list at the destination database. For example, a rule-based transformation may take three columns that store street, state, and postal code data from a source database and combine the data into a single address column at a destination database.

Also, in some cases, no conditional supplemental log group is required even if there is more than one column in a column list. For example, an apply handler or rule-based transformation may separate one address column from the source database into multiple columns that are in a column list at the destination database. A rule-based transformation may take an address that includes street, state, and postal code data in one address column at a source database and separate the data into three columns at a destination database.

Note: Prebuilt update conflict handlers do not support LOB columns. Therefore, you should not include LOB columns in the `column_list` parameter when running the procedure `SET_UPDATE_CONFLICT_HANDLER`.

See Also: ["Supplemental Logging in a Streams Environment"](#) on page 2-11

Resolution Columns

The **resolution column** is the column used to identify an update conflict handler. If you use a `MAXIMUM` or `MINIMUM` prebuilt update conflict handler, then the **resolution column** is also the column used to resolve the conflict. The resolution column must be one of the columns in the column list for the handler.

For example, if the `salary` column in the `hr.employees` table is specified as the resolution column for a maximum or minimum conflict handler, then the `salary` column is evaluated to determine whether column list values in the row LCR are applied or the destination database values for the column list are retained.

In either of the following situations involving a resolution column for a conflict, the apply process moves the transaction containing the row LCR that caused the conflict to an exception queue, if the error handler cannot resolve the problem. In these cases, the conflict cannot be resolved and the values of the columns at the destination database remain unchanged:

- The new LCR value and the destination row value for the resolution column are the same (for example, if the resolution column was not the column causing the conflict).
- Either the new LCR value of the resolution column or the current value of the resolution column at the destination database is `NULL`.

Note: Although the resolution column is not used for `OVERWRITE` and `DISCARD` conflict handlers, a resolution column must be specified for these conflict handlers.

Data Convergence

When you share data between multiple databases, and you want the data to be the same at all of these databases, then make sure you use conflict resolution handlers that cause the data to converge at all databases. If you allow changes to shared data at all of your databases, then data convergence for a table is possible only if all databases that are sharing data capture changes to the shared data and propagate these changes to all of the other databases that are sharing the data.

In such an environment, the `MAXIMUM` conflict resolution method can guarantee convergence only if the values in the resolution column are always increasing. A time-based resolution column meets this requirement, as long as successive timestamps on a row are distinct. The `MINIMUM` conflict resolution method can guarantee convergence in such an environment only if the values in the resolution column are always decreasing.

Custom Conflict Handlers

You can create a PL/SQL procedure to use as a custom conflict handler. You use the `SET_DML_HANDLER` procedure in the `DBMS_APPLY_ADM` package to designate one or more custom conflict handlers for a particular table. Specifically, you set the following parameters when you run this procedure to specify a custom conflict handler:

- Set the `object_name` parameter to the fully qualified name of the table for which you want to perform conflict resolution.
- Set the `object_type` parameter to `TABLE`.
- Set the `operation_name` parameter to the type of operation for which the custom conflict handler is called. The possible operations are the following: `INSERT`, `UPDATE`, `DELETE`, and `LOB_UPDATE`.
- If you want an error handler to perform conflict resolution when an error is raised, then set the `error_handler` parameter to `true`. Or, if you want to include conflict resolution in your DML handler, then set the `error_handler` parameter to `false`.

If you specify `false` for this parameter, then, when you execute a row LCR using the `EXECUTE` member procedure for the LCR, the conflict resolution within the DML handler is performed for the specified object and operation(s).

- Specify the procedure to resolve a conflict by setting the `user_procedure` parameter. This user procedure is called to resolve any conflicts on the specified table resulting from the specified type of operation.

If the custom conflict handler cannot resolve the conflict, then the apply process moves the transaction containing the conflict to an exception queue and does not apply the transaction.

If both a prebuilt update conflict handler and a custom conflict handler exist for a particular object, then the prebuilt update conflict handler is invoked only if both of the following conditions are met:

- The custom conflict handler executes the row LCR using the `EXECUTE` member procedure for the LCR.
- The `conflict_resolution` parameter in the `EXECUTE` member procedure for the row LCR is set to `true`.

See Also:

- ["Handlers and Row LCR Processing"](#) on page 4-16 for more information about you update conflict handlers interact with DML handlers and error handlers
- ["Managing a DML Handler"](#) on page 14-14
- ["Managing an Error Handler"](#) on page 14-21
- *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about the `SET_DML_HANDLER` procedure

Streams Tags

This chapter explains the concepts related to Streams tags.

This chapter contains these topics:

- [Introduction to Tags](#)
- [Tags and Rules Created by the DBMS_STREAMS_ADM Package](#)
- [Tags and an Apply Process](#)
- [Avoid Change Cycling with Tags](#)

See Also: ["Managing Streams Tags"](#) on page 16-26

Introduction to 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.

You can configure how tag values are interpreted. 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). Tags may be used for other LCR tracking purposes as well. You can also use tags to specify the set of destination databases for each LCR.

You can control the value of the tags generated in the redo log in the following ways:

- Use the `DBMS_STREAMS.SET_TAG` procedure to specify the value of the redo tags generated in the current session. When a database change is made in the session, the tag becomes part of the redo entry that records the change. Different sessions can have the same tag setting or different tag settings.
- Use the `CREATE_APPLY` or `ALTER_APPLY` procedure in the `DBMS_APPLY_ADM` package to control the value of the redo tags generated when an apply process runs. All sessions coordinated by the apply process coordinator use this tag setting. By default, redo entries generated by an apply process have a tag value that is the hexadecimal equivalent of '00' (double zero).

These tags become part of the LCRs captured by a capture process retrieving changes from the redo log. Based on the rules in the rule set for the capture process, the tag value in the redo entry for a change may determine whether or not the change is captured.

Similarly, once a tag is part of an LCR, the value of the tag may determine whether a propagation propagates the LCR and whether an apply process applies the LCR. The behavior of a transformation, DML handler, or error handler can also depend on the value of the tag. In addition, you can set the tag value for an existing LCR using the `SET_TAG` member procedure for the LCR. For example, you may set a tag in an LCR during a transformation.

See Also: *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about the `SET_TAG` member procedure for LCRs

Tags and Rules Created by the DBMS_STREAMS_ADM Package

When you use a procedure in the DBMS_STREAMS_ADM package to create rules, by default each rule contains a condition that evaluates to TRUE only if the tag is NULL. In DML rules, the condition is the following:

```
:dml.is_null_tag()='Y'
```

In DDL rules, the condition is the following:

```
:ddl.is_null_tag()='Y'
```

Consider a rule set with a single rule and assume the rule contains such a condition. In this case, Streams capture processes, propagations, and apply processes behave in the following way:

- A capture process captures a change only if the tag in the redo log for the change is NULL and the rest of the rule conditions evaluate to TRUE for the change.
- A propagation propagates an event containing an LCR only if the tag in the LCR is NULL and the rest of the rule conditions evaluate to TRUE for the LCR.
- An apply process applies an event containing an LCR only if the tag in the LCR is NULL and the rest of the rule conditions evaluate to TRUE for the LCR.

Specifically, the following procedures in the DBMS_STREAMS_ADM package create rules that contain one of these conditions by default:

- ADD_GLOBAL_PROPAGATION_RULES
- ADD_GLOBAL_RULES
- ADD_SCHEMA_PROPAGATION_RULES
- ADD_SCHEMA_RULES
- ADD_SUBSET_RULES
- ADD_TABLE_PROPAGATION_RULES
- ADD_TABLE_RULES

If you do not want the created rules to contain such a condition, then set the `include_tagged_lcr` parameter to `true` when you run these procedures. This setting results in no conditions relating to tags in the rules. Therefore, rule evaluation of the LCR does not depend on the value of the tag.

For example, consider a table-level rule that evaluates to TRUE for all DML changes to the `hr.locations` table that originated at the `dbs1.net` source database.

Assume the `ADD_TABLE_RULES` procedure is run to generate this rule:

```
BEGIN DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name           => 'hr.locations',
    streams_type         => 'capture',
    streams_name         => 'capture',
    queue_name           => 'streams_queue',
    include_tagged_lcr   => false, -- Note parameter setting
    source_database      => 'dbs1.net',
    include_dml          => true,
    include_ddl          => false);
END;
/
```

Notice that the `include_tagged_lcr` parameter is set to `false`, which is the default. The `ADD_TABLE_RULES` procedure generates 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' )
```

If a capture process uses a rule set that contains this rule, then the rule evaluates to FALSE if the tag for a change in a redo entry is a non-NULL value, such as '0' or '1'. So, if a redo entry contains a row change to the `hr.locations` table, then the change is captured only if the tag for the redo entry is NULL.

However, suppose the `include_tagged_lcr` parameter is set to `true` when `ADD_TABLE_RULES` is run:

```
BEGIN DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name           => 'hr.locations',
    streams_type         => 'capture',
    streams_name         => 'capture',
    queue_name           => 'streams_queue',
    include_tagged_lcr   => true, -- Note parameter setting
    source_database      => 'dbs1.net',
    include_dml          => true,
    include_ddl          => false);
END;
/
```

In this case, the `ADD_TABLE_RULES` procedure generates a rule with a rule condition similar to the following:

```
((:dml.get_object_owner() = 'HR' and :dml.get_object_name() = 'LOCATIONS'))  
and :dml.get_source_database_name() = 'DBS1.NET' )
```

Notice that there is no condition relating to the tag. If a capture process uses a rule set that contains this rule, then the rule evaluates to `TRUE` if the tag in a redo entry for a DML change to the `hr.locations` table is a non-NULL value, such as `'0'` or `'1'`. The rule also evaluates to `TRUE` if the tag is `NULL`. So, if a redo entry contains a DML change to the `hr.locations` table, then the change is captured regardless of the value for the tag.

If you want to modify the `is_null_tag` condition in a system-created rule, then you should use an appropriate procedure in the `DBMS_STREAMS_ADM` package to create a new rule that is the same as the rule you want to modify, except for the `is_null_tag` condition. Then, use the `REMOVE_RULE` procedure in the `DBMS_STREAMS_ADM` package to remove the old rule from the appropriate rule set.

If you created a rule with the `DBMS_RULE_ADM` package, then you can add, remove, or modify the `is_null_tag` condition in the rule by using the `ALTER_RULE` procedure in this package.

If you are using global rules to capture and apply DDL changes for an entire database, then online backup statements will be captured, propagated, and applied by default. Typically, database administrators do not want to replicate online backup statements. Instead, they only want them to run at the database where they are executed originally. To avoid replicating online backup statements, you can use one of the following strategies:

- Include one or more calls to the `DBMS_STREAMS.SET_TAG` procedure in your online backup procedures, and set the session tag to a value that will cause the online backup statements to be ignored by a capture process.
- Use a DDL handler for an apply process to avoid applying the online backup statements.

See Also:

- [Chapter 6, "How Rules Are Used In Streams"](#) for examples of rules generated by the procedures in the `DBMS_STREAMS_ADM` package
- *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about the `DBMS_STREAMS_ADM` package and the `DBMS_RULE_ADM.ALTER_RULE` procedure
- ["Setting the Tag Values Generated by the Current Session"](#) on page 16-26 for more information about the `SET_TAG` procedure

Tags and an Apply Process

An apply process generates entries in the redo log of a destination database when it applies DML or DDL changes. For example, if the apply process applies a change that updates a row in a table, then that change is recorded in the redo log at the destination database. You can control the tags in these redo entries by setting the `apply_tag` parameter in the `CREATE_APPLY` or `ALTER_APPLY` procedure in the `DBMS_APPLY_ADM` package. For example, an apply process may generate redo tags that are equivalent to the hexadecimal value of '0' (zero) or '1'.

The default tag value generated in the redo log by an apply process is '00' (double zero). This value is the default tag value for an apply process if you use a procedure in the `DBMS_STREAMS_ADM` package or the `CREATE_APPLY` procedure in the `DBMS_APPLY_ADM` package to create an apply process. There is nothing special about this value beyond the fact that it is a non-NULL value. The fact that it is a non-NULL value is important because rules created by the `DBMS_STREAMS_ADM` package by default contain a condition that evaluates to `TRUE` only if the tag is `NULL` in a redo entry or LCR. You can alter the tag value for an existing apply process using the `ALTER_APPLY` procedure in the `DBMS_APPLY_ADM` package.

If a DML handler, DDL handler, or message handler calls the `SET_TAG` procedure in the `DBMS_STREAMS` package, then any subsequent redo entries generated by the handler will include the tag specified in the `SET_TAG` call, even if the tag for the apply process is different. When the handler exits, any subsequent redo entries generated by the apply process have the tag specified for the apply process.

See Also:

- [Chapter 4, "Streams Apply Process"](#) for more information about the apply process
- ["Tags and Rules Created by the DBMS_STREAMS_ADM Package"](#) on page 8-3 for more information about the default tag condition in Streams rules
- ["Setting the Tag Values Generated by an Apply Process"](#) on page 16-28
- ["Event Processing Options"](#) on page 4-4 for more information about DML handlers, DDL handlers, and message handlers
- ["Setting the Tag Values Generated by the Current Session"](#) on page 16-26 for more information about the `SET_TAG` procedure
- *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about the `DBMS_STREAMS_ADM` package and the `DBMS_APPLY_ADM` package

Avoid Change Cycling with Tags

In a Streams environment that includes more than one database sharing data bidirectionally, you can use tags to avoid **change cycling**. Change cycling means sending a change back to the database where it originated. Typically, change cycling should be avoided because it can result in each change going through endless loops back to the database where it originated. Such loops can result in unintended data in the database and tax the networking and computer resources of an environment. By default, Streams is designed to avoid change cycling.

Using tags and appropriate rules for Streams capture processes, propagations, and apply processes, you can avoid such change cycles. The following sections describe various Streams environments and how tags and rules can be used to avoid change cycling in these environments:

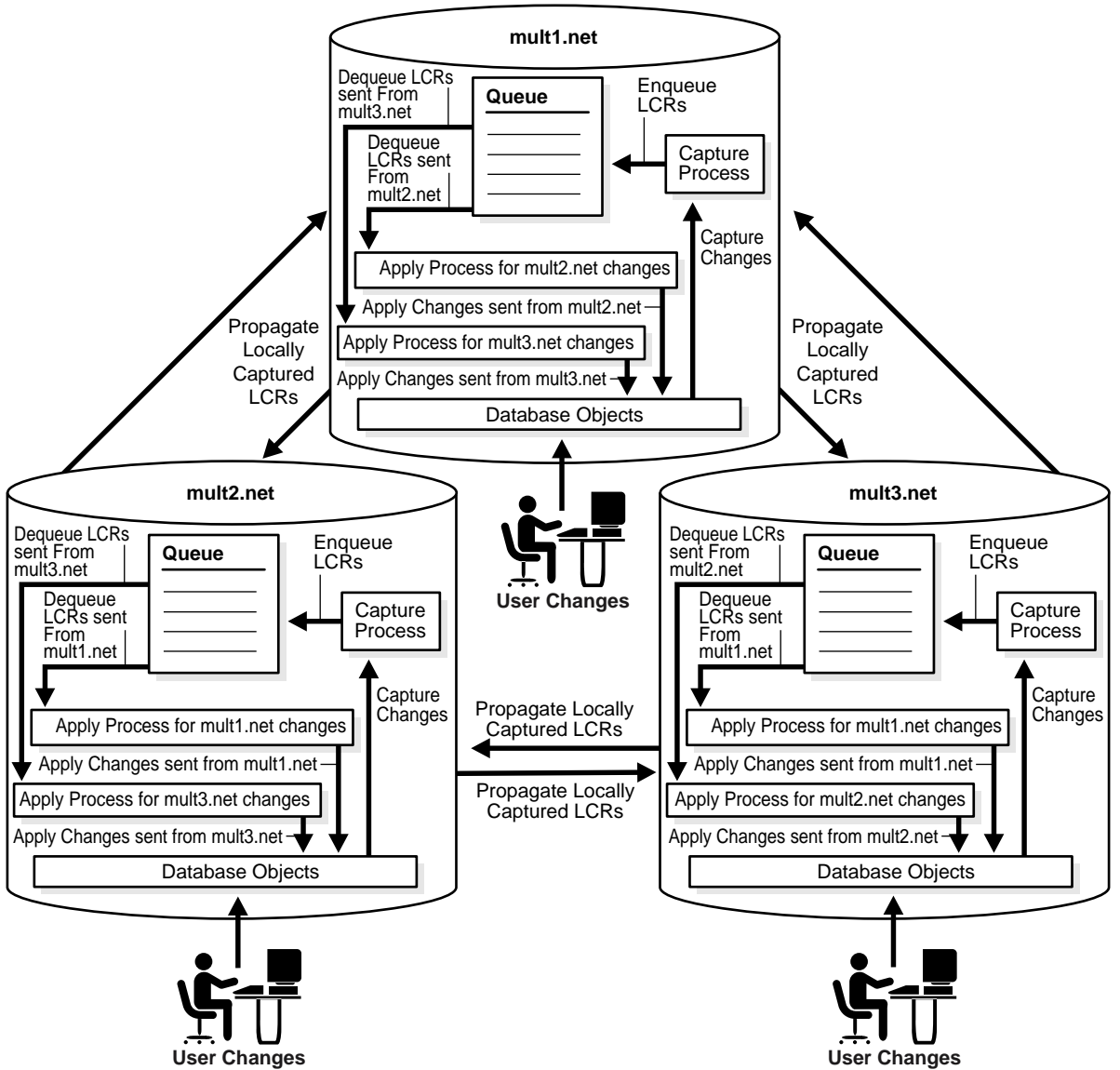
- [Each Databases Is a Source and Destination Database for Shared Data](#)
- [Primary Database Sharing Data with Several Secondary Databases](#)
- [Primary Database Sharing Data with Several Extended Secondary Databases](#)

Each Databases Is a Source and Destination Database for Shared Data

This scenario involves a Streams environment in which each database is a source database for every other database, and each database is a destination database of every other database. Each database communicates directly with every other database.

For example, consider an environment that replicates the database objects and data in the `hr` schema between three Oracle databases: `mult1.net`, `mult2.net`, and `mult3.net`. DML and DDL changes made to tables in the `hr` schema are captured at all three databases in the environment and propagated to each of the other databases in the environment, where changes are applied. [Figure 8-1](#) illustrates an example environment in which each database is a source database.

Figure 8–1 Each Database Is a Source and Destination Database



You can avoid change cycles by configuring such an environment in the following way:

- Configure one apply process at each database to generate non-NULL redo tags for changes from each source database. If you use a procedure in the `DBMS_STREAMS_ADM` package to create an apply process, then the apply process generates non-NULL tags with a value of '00' in the redo log by default. In this case, no further action is required for the apply process to generate non-NULL tags.

If you use the `CREATE_APPLY` procedure in the `DBMS_APPLY_ADM` package to create an apply process, then do not set the `apply_tag` parameter. Again, the apply process generates non-NULL tags with a value of '00' in the redo log by default, and no further action is required.

- Configure the capture process at each database to capture changes only if the tag in the redo entry for the change is NULL. You do this by ensuring that each DML rule in the rule set used by the capture process has the following condition:

```
:dml.is_null_tag()='Y'
```

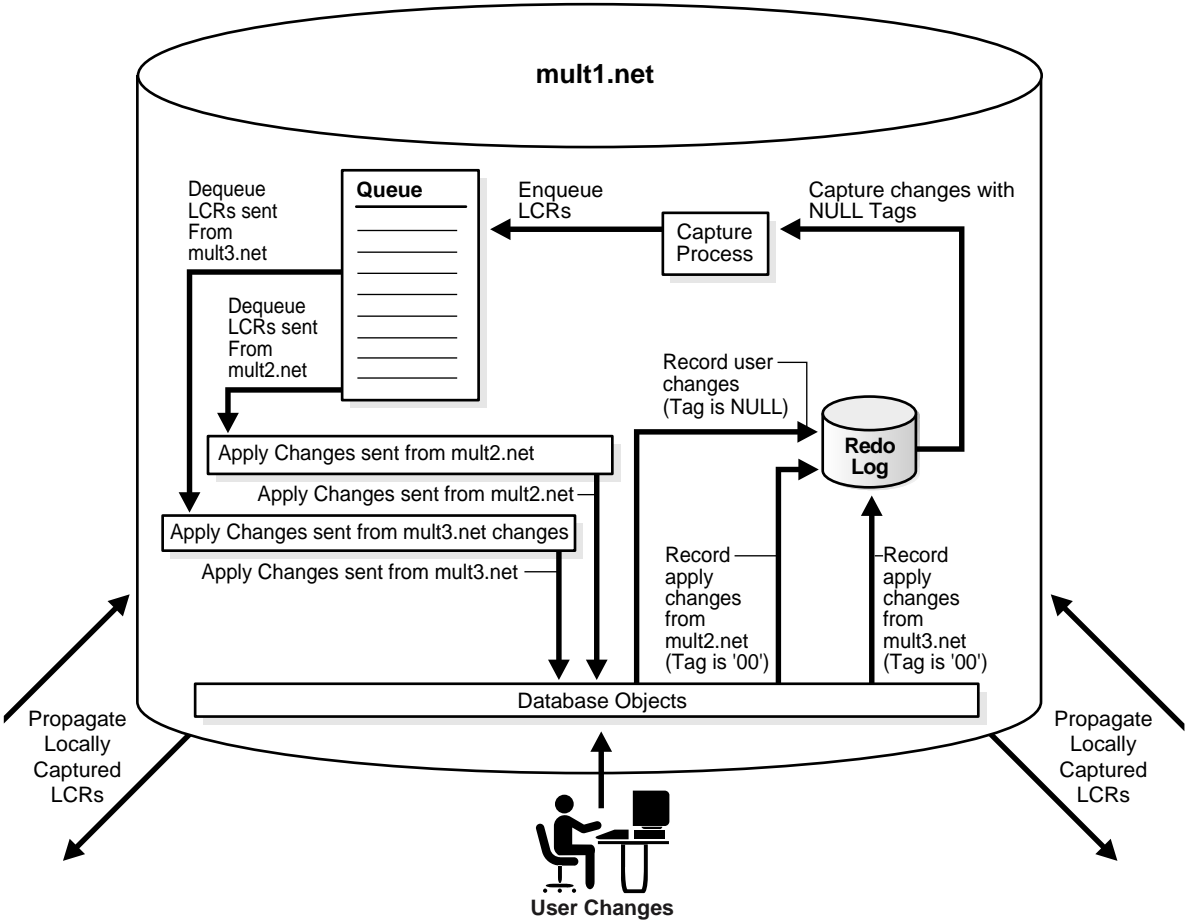
Each DDL rule should have the following condition:

```
:ddl.is_null_tag()='Y'
```

These rule conditions indicate that the capture process captures a change only if the tag for the change is NULL. If you use the `DBMS_STREAMS_ADM` package to generate rules, then each rule has one of these conditions by default.

This configuration prevents change cycling because all of the changes applied by the apply processes are never recaptured (they were captured originally at the source databases). Each database sends all of its changes to the `hr` schema to every other database. So, in this environment, no changes are lost, and all databases are synchronized. [Figure 8-2](#) illustrates how tags can be used in a database in a multiple source environment.

Figure 8-2 Tag Use When Each Database Is a Source and Destination Database



See Also: [Chapter 23, "Multiple Source Replication Example"](#) illustrates this example

Primary Database Sharing Data with Several Secondary Databases

This scenario involves a Streams environment in which one database is the primary database, and this primary database shares data with several secondary databases. The secondary databases share data only with the primary database. The secondary databases do not share data directly with each other, but, instead, share data indirectly with each other through the primary database. This type of environment is sometimes called a "hub and spoke" environment, with the primary database being the hub and the secondary databases being the spokes.

In such an environment, changes are captured, propagated, and applied in the following way:

- The primary database captures local changes to the shared data and propagates these changes to all secondary databases, where these changes are applied at each secondary database locally.
- Each secondary database captures local changes to the shared data and propagates these changes to the primary database only, where these changes are applied at the primary database locally.
- The primary database applies changes from each secondary database locally. Then, these changes are captured at the primary database and propagated to all secondary databases, except for the one at which the change originated. Each secondary database applies the changes from the other secondary databases locally, after they have gone through the primary database. This configuration is an example of apply forwarding.

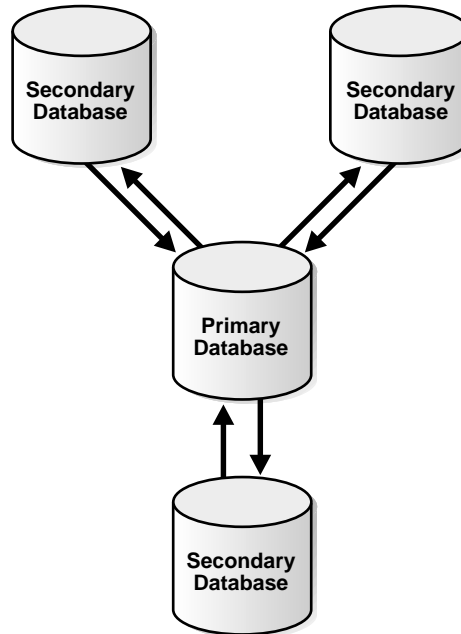
An alternate scenario may use queue forwarding. If this environment used queue forwarding, then changes from secondary databases that are applied at the primary database are not captured at the primary database. Instead, these changes are forwarded from the queue at the primary database to all secondary databases, except for the one at which the change originated.

See Also: ["Directed Networks"](#) on page 3-7 for more information about apply forwarding and queue forwarding

For example, consider an environment that replicates the database objects and data in the `hr` schema between one primary database named `ps1.net` and three secondary databases named `ps2.net`, `ps3.net`, and `ps4.net`. DML and DDL changes made to tables in the `hr` schema are captured at the primary database and at the three secondary databases in the environment. Then, these changes are propagated and applied as described previously. The environment uses apply forwarding, not queue forwarding, to share data between the secondary databases

through the primary database. [Figure 8-3](#) illustrates an example environment which has one primary database and multiple secondary databases.

Figure 8-3 Primary Database Sharing Data with Several Secondary Databases



You can avoid change cycles by configuring the environment in the following way:

- Configure each apply process at the primary database `ps1.net` to generate non-NULL redo tags that indicate the site from which it is receiving changes. In this environment, the primary database has at least one apply process for each secondary database from which it receives changes. For example, if an apply process at the primary database receives changes from the `ps2.net` secondary site, then this apply process may generate a raw value that is equivalent to the hexadecimal value '2' for all changes it applies. You do this by setting the `apply_tag` parameter in the `CREATE_APPLY` or `ALTER_APPLY` procedure in the `DBMS_APPLY_ADM` package to the non-NULL value.

For example, run the following procedure to create an apply process that generates redo entries with tags that are equivalent to the hexadecimal value '2':

```
BEGIN
  DBMS_APPLY_ADM.CREATE_APPLY(
    queue_name      => 'strmadmin.streams_queue',
    apply_name      => 'apply_ps2',
    rule_set_name   => 'strmadmin.apply_rules_ps2',
    apply_tag       => HEXTORAW('2'),
    apply_captured  => true);
END;
/
```

- Configure the apply process at each secondary database to generate non-NULL redo tags. The exact value of the tags is irrelevant as long as it is non-NULL. In this environment, each secondary database has one apply process that applies changes from the primary database.

If you use a procedure in the `DBMS_STREAMS_ADM` package to create an apply process, then the apply process generates non-NULL tags with a value of '00' in the redo log by default. In this case, no further action is required for the apply process to generate non-NULL tags.

For example, assuming no apply processes exist at the secondary databases, run the `ADD_SCHEMA_RULES` procedure in the `DBMS_STREAMS_ADM` package at each secondary database to create an apply process that generates non-NULL redo entries with tags that are equivalent to the hexadecimal value '00':

```
BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
    schema_name     => 'hr',
    streams_type    => 'apply',
    streams_name    => 'apply',
    queue_name      => 'strmadmin.streams_queue',
    include_dml     => true,
    include_ddl     => true,
    source_database => 'ps1.net');
END;
/
```

- Configure the capture process at the primary database to capture changes to the shared data regardless of the tags. You do this by setting the `include_tagged_lcr` parameter to `true` when you run one of the procedures that generate capture rules in the `DBMS_STREAMS_ADM` package. If you use the `DBMS_RULE_ADM` package to create rules for the capture process at the primary database, then make sure the rules do not contain `is_null_tag` conditions, because these conditions involve tags in the redo log.

For example, run the following procedure at the primary database to produce one DML capture process rule and one DDL capture process rule that each have a condition that evaluates to `TRUE` for changes in the `hr` schema, regardless of the tag for the change:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
    schema_name      => 'hr',
    streams_type     => 'capture',
    streams_name     => 'capture',
    queue_name       => 'strmadmin.streams_queue',
    include_tagged_lcr => true, -- Note parameter setting
    include_dml      => true,
    include_ddl      => true);
END;
/
```

- Configure the capture process at each secondary database to capture changes only if the tag in the redo entry for the change is `NULL`. You do this by ensuring that each DML rule in the rule set used by the capture process at the secondary database has the following condition:

```
:dml.is_null_tag()='Y'
```

DDL rules should have the following condition:

```
:ddl.is_null_tag()='Y'
```

These rules indicate that the capture process captures a change only if the tag for the change is `NULL`. If you use the `DBMS_STREAMS_ADM` package to generate rules, then each rule has one of these conditions by default. If you use the `DBMS_RULE_ADM` package to create rules for the capture process at a secondary database, then make sure each rule contains one of these conditions.

- Configure one propagation from the queue at the primary database to the queue at each secondary database. Each propagation should use a rule set with rules that instruct the propagation to propagate all LCRs in the queue at the primary database to the queue at the secondary database, except for changes that originated at the secondary database.

For example, if a propagation propagates changes to the secondary database `ps2.net`, whose tags are equivalent to the hexadecimal value '2', then the rules for the propagation should propagate all LCRs relating to the `hr` schema to the secondary database, except for LCRs with a tag of '2'. For row LCRs, such rules should include the following condition:

```
:dml.get_tag() !=HEXTORAW('2')
```

For DDL LCRs, such rules should include the following condition:

```
:ddl.get_tag() !=HEXTORAW('2')
```

You can use the `CREATE_RULE` procedure in the `DBMS_RULE_ADM` package to create rules with these conditions.

- Configure one propagation from the queue at each secondary database to the queue at the primary database. A queue at one of the secondary databases contains only local changes made by user sessions and applications at the secondary database, not changes made by an apply process. Therefore, no further configuration is necessary for these propagations.

This configuration prevents change cycling in the following way:

- Changes that originated at a secondary database are never propagated back to that secondary database.
- Changes that originated at the primary database are never propagated back to the primary database.
- All changes made to the shared data at any database in the environment are propagated to every other database in the environment.

So, in this environment, no changes are lost, and all databases are synchronized.

Figure 8-4 illustrates how tags are used at the primary database ps1.net.

Figure 8-4 Tags Used at the Primary Database

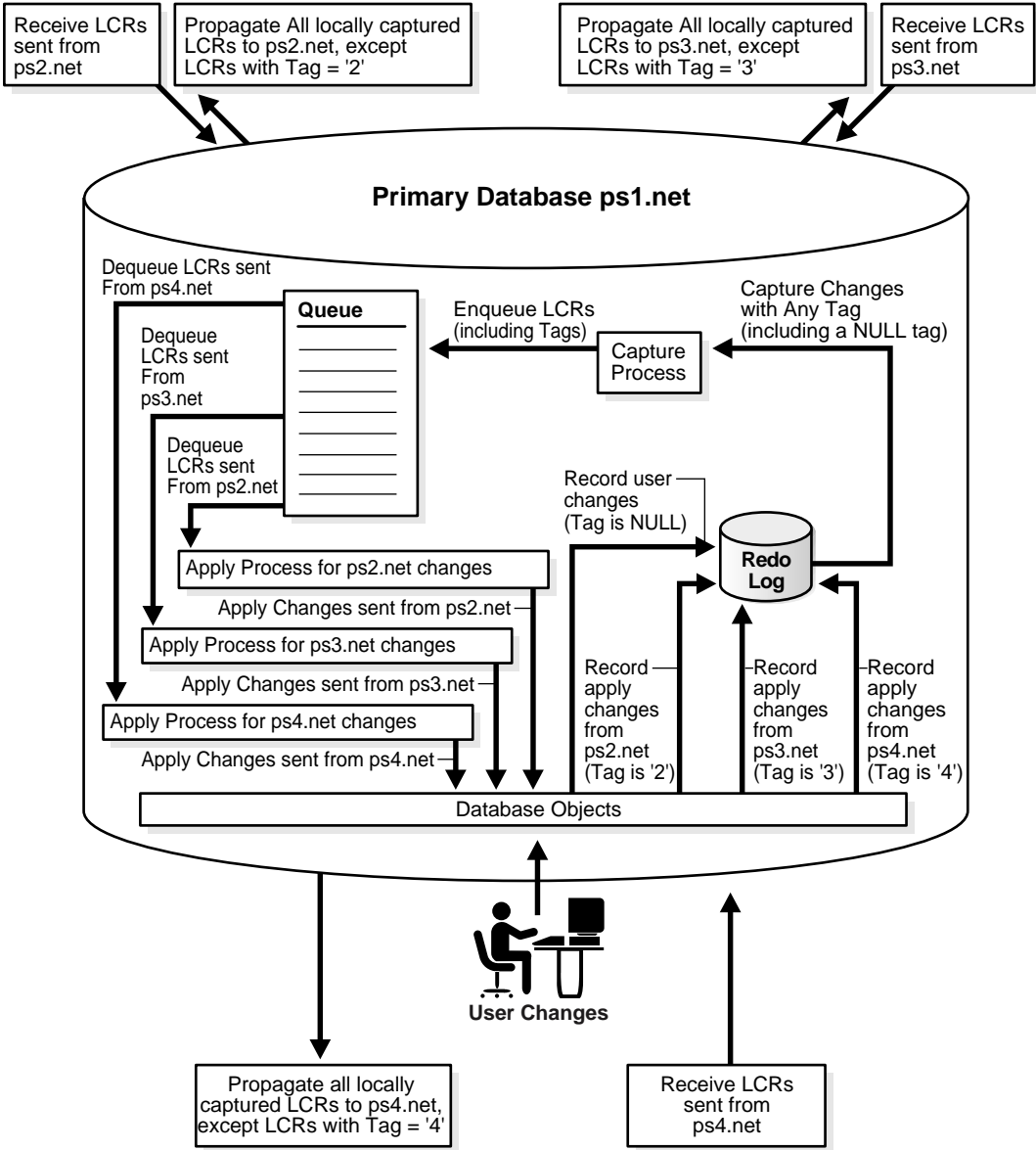
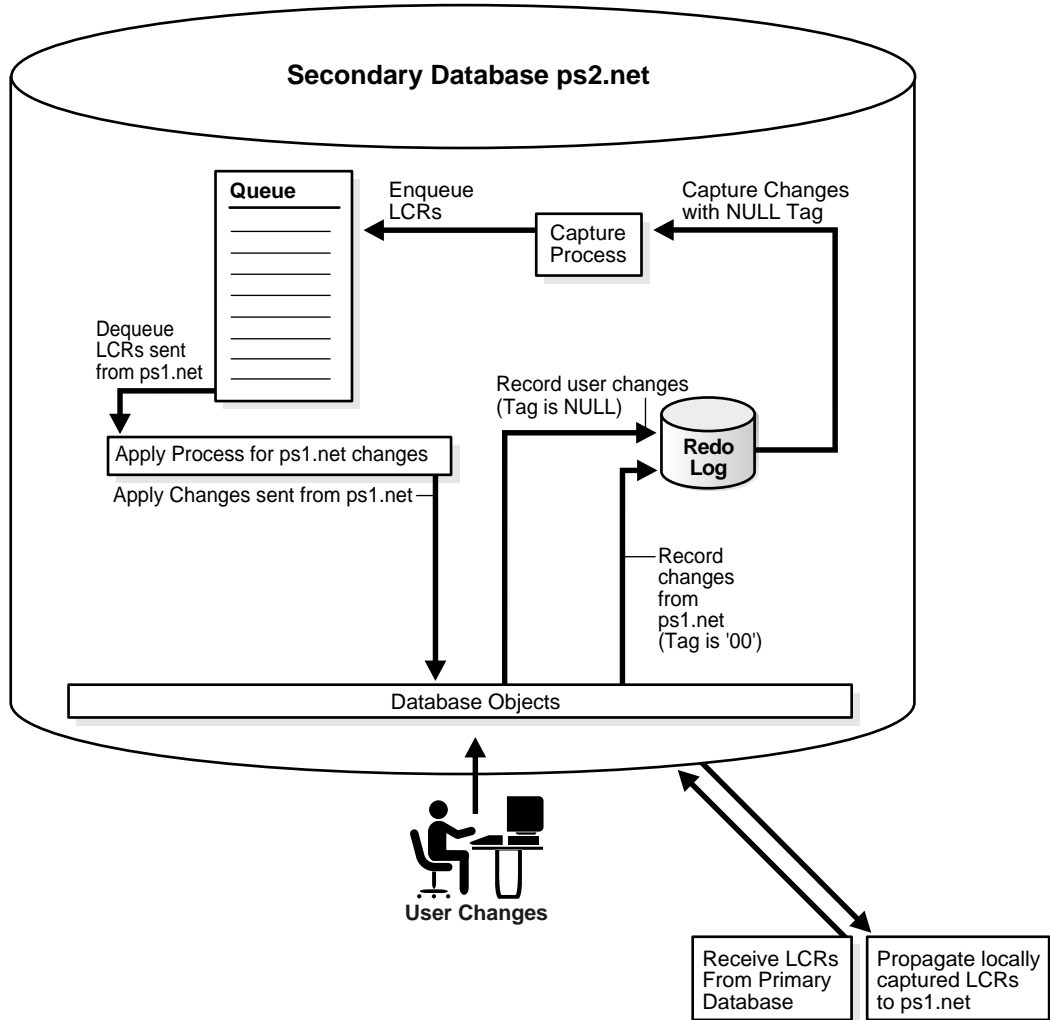


Figure 8-5 illustrates how tags are used at one of the secondary databases (ps2.net).

Figure 8-5 Tags Used at a Secondary Database

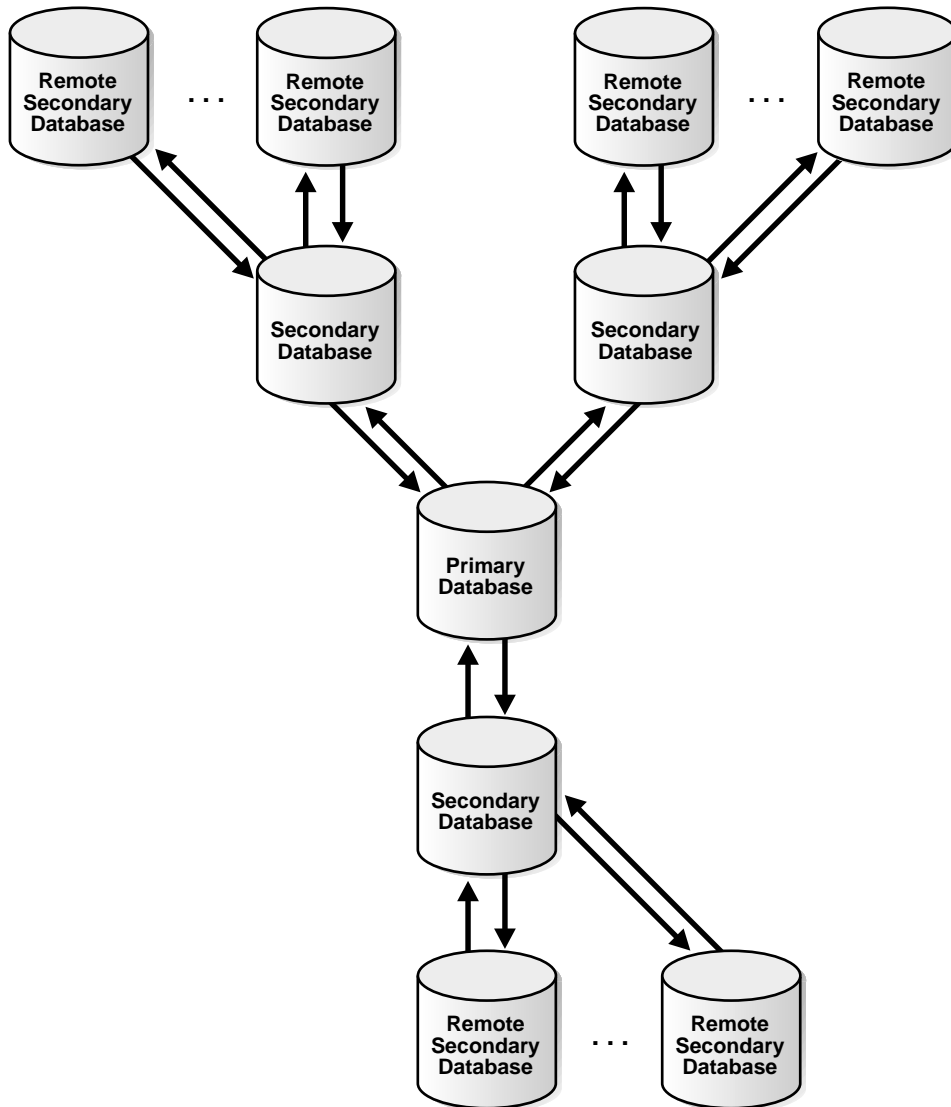


Primary Database Sharing Data with Several Extended Secondary Databases

In this environment, one primary database shares data with several secondary databases, but the secondary databases have other secondary databases connected to them, which will be called *remote secondary* databases. This environment is an extension of the environment described in "[Primary Database Sharing Data with Several Secondary Databases](#)" on page 8-12.

A remote secondary database does not share data directly with the primary database, but instead shares data indirectly with the primary database through a secondary database. So, the shared data exists at the primary database, at each secondary database, and at each remote secondary database. Changes made at any of these databases are captured and propagated to all of the other databases. [Figure 8-6](#) illustrates an environment with one primary database and multiple extended secondary databases.

Figure 8–6 Primary Database and Several Extended Secondary Databases



In such an environment, you can avoid change cycling in the following way:

- Configure the primary database in the same way that it is configured in the example described in "[Primary Database Sharing Data with Several Secondary Databases](#)" on page 8-12.
- Configure each remote secondary database similar to the way that each secondary database is configured in the example described in "[Primary Database Sharing Data with Several Secondary Databases](#)" on page 8-12. The only difference is that the remote secondary databases share data directly with secondary databases, not the primary database.
- At each secondary database, configure one apply process to apply changes from the primary database with a redo tag value that is equivalent to the hexadecimal value '00'. This value is the default tag value for an apply process.
- At each secondary database, configure one apply process to apply changes from each of its remote secondary databases with a redo tag value that is unique for the remote secondary database.
- Configure the capture process at each secondary database to capture all changes to the shared data in the redo log, regardless of the tag value for the changes.
- Configure one propagation from the queue at each secondary database to the queue at the primary database. The propagation should use a rule set with rules that instruct the propagation to propagate all LCRs in the queue at the secondary database to the queue at the primary database, except for changes that originated at the primary database. You do this by adding a condition to the rules that evaluates to `TRUE` only if the tag in the LCR does not equal '00'. For example, enter a condition similar to the following for row LCRs:

```
:dml.get_tag() !=HEXTORAW('00')
```

- Configure one propagation from the queue at each secondary database to the queue at each remote secondary database. Each propagation should use a rule set with rules that instruct the propagation to propagate all LCRs in the queue at the secondary database to the queue at the remote secondary database, except for changes that originated at the remote secondary database. You do this by adding a condition to the rules that evaluates to `TRUE` only if the tag in the LCR does not equal the tag value for the remote secondary database. For example, if the tag value of a remote secondary database is equivalent to the hexadecimal value `'19'`, then enter a condition similar to the following for row LCRs:

```
:dm1.get_tag() !=HEXTORAW('19')
```

By configuring the environment in this way, you prevent change cycling, and no changes originating at any database are lost.

Streams Heterogeneous Information Sharing

This chapter explains concepts relating to Streams support for information sharing between Oracle databases and non-Oracle databases.

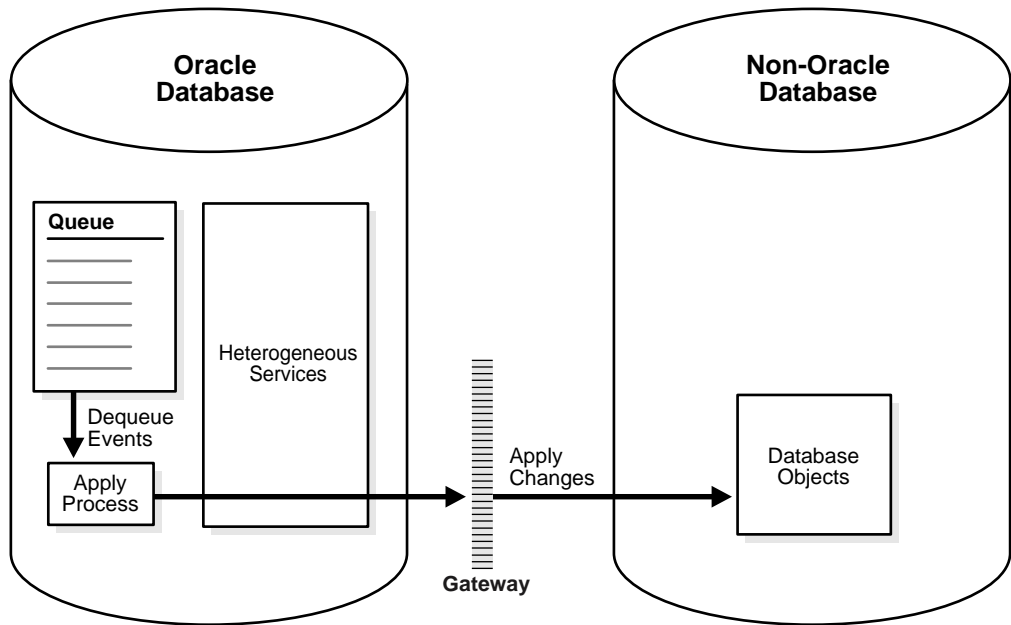
This chapter contains these topics:

- [Oracle to Non-Oracle Data Sharing with Streams](#)
- [Non-Oracle to Oracle Data Sharing with Streams](#)
- [Non-Oracle to Non-Oracle Data Sharing with Streams](#)

Oracle to Non-Oracle Data Sharing with Streams

To share DML changes from an Oracle source database to 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 applies the changes to the non-Oracle database across a network connection through a gateway. [Figure 9-1](#) shows an Oracle databases sharing data with a non-Oracle database.

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



Change Capture and Staging in an Oracle to Non-Oracle Environment

In an Oracle to non-Oracle environment, the capture process functions the same way as it would in an Oracle-only environment. That is, it finds changes in the redo log, captures them based on capture process rules, and enqueues the captured changes as logical change records (LCRs) in a `SYS.AnyData` queue. In addition, a single capture process may capture changes that will be applied at both Oracle and non-Oracle databases.

Similarly, the `SYS.AnyData` queue that stages the captured LCRs functions the same way as it would in an Oracle-only environment, and you can propagate LCRs to any number of intermediate queues in Oracle databases before they are applied at a non-Oracle database.

See Also:

- [Chapter 2, "Streams Capture Process"](#)
- [Chapter 3, "Streams Staging and Propagation"](#)

Change Apply in an Oracle to Non-Oracle Environment

An apply process running in an Oracle database uses Heterogeneous Services and a gateway to apply changes encapsulated in LCRs directly to database objects in a non-Oracle database. The LCRs are not propagated to a queue in the non-Oracle database, as they would be in an Oracle-only Streams environment. Instead, the apply process applies the changes directly through a database link to the non-Oracle database.

See Also: [Chapter 4, "Streams Apply Process"](#) for detailed information about the apply process

Apply Process Configuration in an Oracle to Non-Oracle Environment

This section describes the configuration of an apply process that will apply changes to a non-Oracle database.

Database Link to the Non-Oracle Database When you create an apply process that will apply changes to a non-Oracle database, you must previously have configured Heterogeneous Services, the gateway, and a database link, which will be used by the apply process to apply the changes to the non-Oracle database. The database link must be created with an explicit `CONNECT TO` clause.

When the database link is created and working properly, create the apply process using the `CREATE_APPLY` procedure in the `DBMS_APPLY_ADM` package and specify the database link for the `apply_database_link` parameter. After you create an apply process, you can use apply process rules to specify which changes are applied at the non-Oracle database.

See Also:

- *Oracle9i Heterogeneous Connectivity Administrator's Guide* for more information about Heterogeneous Services and gateways
- *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about the procedures in the `DBMS_APPLY_ADM` package
- [Chapter 6, "How Rules Are Used In Streams"](#) for information about specifying apply process rules

Substitute Key Columns in an Oracle to Non-Oracle Heterogeneous Environment If you use substitute key columns for any of the tables at the non-Oracle database, then specify the database link to the non-Oracle database when you run the `SET_KEY_COLUMNS` procedure in the `DBMS_APPLY_ADM` package.

See Also:

- ["Substitute Key Columns"](#) on page 4-11
- ["Managing the Substitute Key Columns for a Table"](#) on page 14-27

Parallelism in an Oracle to Non-Oracle Heterogeneous Environment You must set the `parallelism_apply` process parameter to 1, the default setting, when an apply process is applying changes to a non-Oracle database. Currently, parallel apply to non-Oracle databases is not supported. However, you may use multiple apply processes to apply changes a non-Oracle database.

DML Handlers in an Oracle to Non-Oracle Heterogeneous Environment If you use a DML handler to process row LCRs for any of the tables at the non-Oracle database, then specify the database link to the non-Oracle database when you run the `SET_DML_HANDLER` procedure in the `DBMS_APPLY_ADM` package.

See Also:

- ["Event Processing Options"](#) on page 4-4
- ["Managing a DML Handler"](#) on page 14-14

Message Handlers in an Oracle to Non-Oracle Heterogeneous Environment If you want to use a message handler to process user-enqueued messages for a non-Oracle database, then, when you run the `CREATE_APPLY` procedure in the `DBMS_APPLY_ADM` package, specify the database link to the non-Oracle database using the `apply_database_link` parameter, and specify the message handler procedure using the `message_handler` parameter.

See Also:

- ["Event Processing Options"](#) on page 4-4
- ["Managing the Message Handler for an Apply Process"](#) on page 14-13

Error and Conflict Handlers in an Oracle to Non-Oracle Heterogeneous Environment Currently, error handlers and conflict handlers are not supported when sharing data from an Oracle database to a non-Oracle database. If an apply error occurs, then the transaction containing the LCR that caused the error is moved into an exception queue in the Oracle database.

Datatypes Applied at Non-Oracle Databases

When applying changes to a non-Oracle database, an apply process applies changes made to columns of only the following datatypes:

- CHAR
- VARCHAR2
- NCHAR
- NVARCHAR2
- NUMBER
- DATE
- RAW
- TIMESTAMP
- TIMESTAMP WITH TIME ZONE
- TIMESTAMP WITH LOCAL TIME ZONE
- INTERVAL YEAR TO MONTH
- INTERVAL DAY TO SECOND

The apply process does not apply changes in columns of the following datatypes to non-Oracle databases: CLOB, NCLOB, BLOB, BFILE, LONG, LONG RAW, ROWID, UROWID, and user-defined types (including object types, REFS, varrays, and nested tables). The apply process raises an error when an LCR contains a datatype that is not listed, and the transaction containing the LCR that caused the error is moved to an exception queue in the Oracle database.

Each transparent gateway may have further limitations regarding datatypes. For a datatype to be supported in an Oracle to non-Oracle environment, the datatype must be supported by both Streams and the gateway being used.

See Also:

- *Oracle9i SQL Reference* for more information about these datatypes
- Your Oracle-supplied gateway-specific documentation for information about transparent gateways

Types of DML Changes Applied at Non-Oracle Databases

When you specify that DML changes made to certain tables should be applied at a non-Oracle database, an apply process can apply only the following types of DML changes:

- INSERT
- UPDATE
- DELETE

Note: The apply process cannot apply DDL changes at non-Oracle databases.

Instantiation in an Oracle to Non-Oracle Environment

Before you start an apply process that applies changes to a non-Oracle database, complete the following steps to instantiate each table at the non-Oracle database:

1. Use the `DBMS_HS_PASSTHROUGH` package or the tools supplied with the non-Oracle database to create the table at the non-Oracle database.

See Also: *Oracle9i Heterogeneous Connectivity Administrator's Guide* and your Oracle supplied gateway-specific documentation for more information about Heterogeneous Services and transparent gateways

2. If the changes that will be shared between the Oracle and non-Oracle database are captured by a capture process at the Oracle database, then prepare all tables that will share data for instantiation.

See Also: ["Preparing Database Objects for Instantiation at a Source Database"](#) on page 12-11

3. Create a PL/SQL procedure (or a C program) that performs the following actions:
 - Gets the current SCN using the `GET_SYSTEM_CHANGE_NUMBER` function in the `DBMS_FLASHBACK` package.
 - Invokes the `ENABLE_AT_SYSTEM_CHANGE_NUMBER` procedure in the `DBMS_FLASHBACK` package to set the current session to the obtained SCN. This action ensures that all fetches are done using the same SCN.

- Populates the table at the non-Oracle site by fetching row by row from the table at the Oracle database and then inserting row by row into the table at the non-Oracle database. All fetches should be done at the SCN obtained using the `GET_SYSTEM_CHANGE_NUMBER` function.

For example, the following PL/SQL procedure gets the flashback SCN, fetches each row in the `hr.regions` table in the current Oracle database, and inserts them into the `hr.regions` table in the `het.net` non-Oracle database. Notice that flashback is disabled before the rows are inserted into the non-Oracle database.

```
SET SERVEROUTPUT ON
CREATE OR REPLACE PROCEDURE insert_reg IS
  CURSOR c1 IS
    SELECT region_id, region_name FROM hr.regions;
  c1_rec c1 % ROWTYPE;
  scn NUMBER;
BEGIN
  scn := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER();
  DBMS_FLASHBACK.ENABLE_AT_SYSTEM_CHANGE_NUMBER(
    query_scn => scn);
  /* Open c1 in flashback mode */
  OPEN c1;
  /* Disable Flashback */
  DBMS_FLASHBACK.DISABLE;
  LOOP
    FETCH c1 INTO c1_rec;
    EXIT WHEN c1%NOTFOUND;
    /*
     Note that all the DML operations inside the loop are performed
     with Flashback disabled
    */
    INSERT INTO hr.regions@het.net VALUES (
      c1_rec.region_id,
      c1_rec.region_name);
  END LOOP;
  COMMIT;
  DBMS_OUTPUT.PUT_LINE('SCN = ' || scn);
  EXCEPTION WHEN OTHERS THEN
    DBMS_FLASHBACK.DISABLE;
    RAISE;
END;
/
```

Make a note of the SCN returned.

Note: The user who creates and runs this procedure must have EXECUTE privilege on the DBMS_FLASHBACK package and all privileges on the tables involved.

4. Specify the SCN you obtained in Step 3 in the SET_TABLE_INSTANTIATION_SCN procedure in the DBMS_APPLY_ADM package to instruct the apply process to skip all LCRs with changes that occurred before the SCN you obtained in Step 3. Make sure you set the apply_database_link parameter to the database link for the remote non-Oracle database.

See Also: ["Setting Instantiation SCNs at a Destination Database"](#) on page 14-35 and *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about the SET_TABLE_INSTANTIATION_SCN procedure in the DBMS_APPLY_ADM package

Transformations in an Oracle to Non-Oracle Environment

In an Oracle to non-Oracle environment, you can specify rule-based transformations during capture or apply the same way as you would in an Oracle-only environment. In addition, if your environment propagates LCRs to one or more intermediate Oracle databases before they are applied at a non-Oracle database, then you can specify a rule-based transformation during propagation from a queue at an Oracle database to another queue at an Oracle database.

See Also: ["Rule-Based Transformations"](#) on page 6-25

Messaging Gateway and Streams

Messaging Gateway is a feature of the Oracle database that provides propagation between Oracle queues and non-Oracle message queuing systems. Messages enqueued into an Oracle queue are automatically propagated to a non-Oracle queue, and the messages enqueued into a non-Oracle queue are automatically propagated to an Oracle queue. It provides guaranteed message delivery to the non-Oracle messaging system and supports the native message format for the non-Oracle messaging system. It also supports specification of user-defined transformations that are invoked while propagating from an Oracle queue to the non-Oracle messaging system or from the non-Oracle messaging system to an Oracle queue.

See Also: *Oracle9i Application Developer's Guide - Advanced Queuing* for more information about the Messaging Gateway

Error Handling in an Oracle to Non-Oracle Environment

If the apply process encounters an unhandled error when it tries to apply an LCR at a non-Oracle database, then the transaction containing the LCR is placed in an exception queue in the Oracle database that is running the apply process. The apply process detects data conflicts in the same way as it does in an Oracle-only environment, but automatic conflict resolution is not supported currently in an Oracle to non-Oracle environment. Therefore, any data conflicts encountered are treated as apply errors.

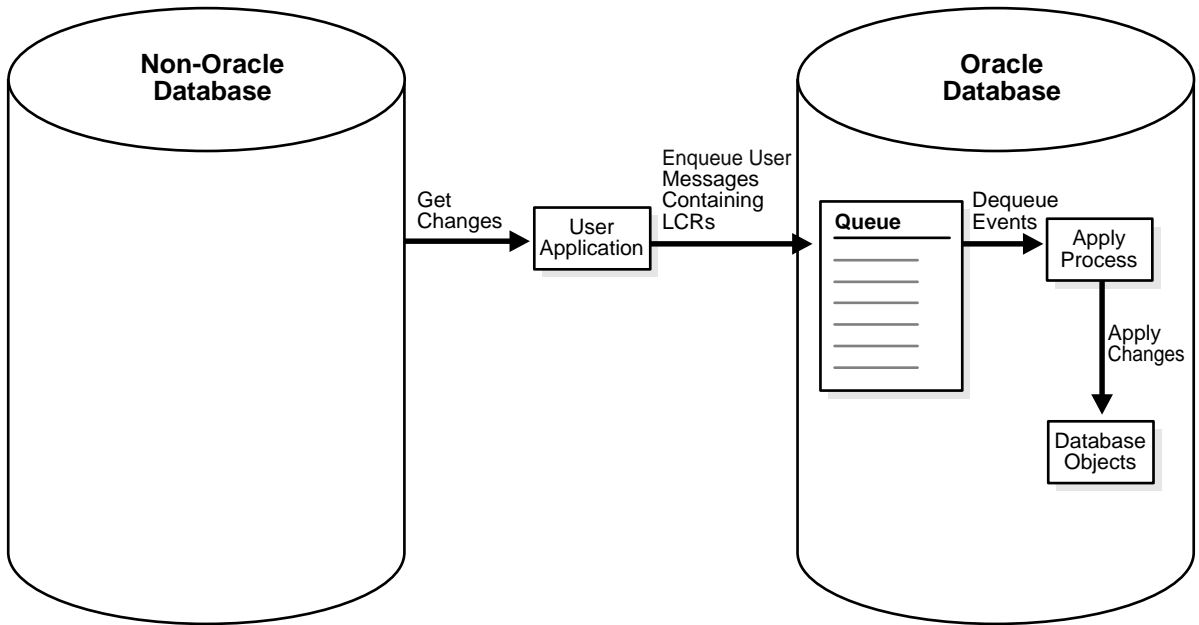
Example Oracle to Non-Oracle Streams Environment

[Chapter 22, "Single Source Heterogeneous Replication Example"](#) contains a detailed example that includes sharing data in an Oracle to non-Oracle Streams environment.

Non-Oracle to Oracle Data Sharing with Streams

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, by using triggers, or by some other method. The application must assemble and order the transactions and must convert each change into a logical change record (LCR). Then, the application must enqueue the LCRs into a queue in an Oracle database using the `DBMS_AQ` package. The application must commit after enqueueing all LCRs in each transaction. [Figure 9-2](#) shows a non-Oracle databases sharing data with an Oracle database.

Figure 9–2 Non-Oracle to Oracle Heterogeneous Data Sharing



Change Capture and Staging in a Non-Oracle to Oracle Environment

Because the custom user application is responsible for assembling changes at the non-Oracle database into LCRs and enqueueing the LCRs into a queue at the Oracle database, the application is completely responsible for change capture. This means that the application must construct LCRs that represent changes at the non-Oracle database and then enqueue these LCRs into the queue at the Oracle database. The application must enqueue transactions serially in the same order as the transactions committed on the non-Oracle source database.

If you want to ensure the same transactional consistency at both the Oracle database where changes are applied and the non-Oracle database where changes originate, then you must use a transactional queue to stage the LCRs at the Oracle database. For example, suppose a single transaction contains three row changes, and the custom application enqueues three row LCRs, one for each change, and then commits. With a transactional queue, a commit is performed by the apply process after the third row LCR, retaining the consistency of the transaction. If you use a nontransactional queue, then a commit is performed for each row LCR by the apply

process. The `SET_UP_QUEUE` procedure in the `DBMS_STREAMS_ADM` package creates a transactional queue automatically.

See Also:

- ["Constructing and Enqueuing LCRs"](#) on page 16-2
- ["SYS.AnyData Queues and User Messages"](#) on page 3-11

Change Apply in a Non-Oracle to Oracle Environment

In a non-Oracle to Oracle environment, the apply process functions the same way as it would in an Oracle-only environment. That is, it dequeues each event from its associated queue based on apply process rules, performs any rule-based transformation, and either sends the event to a handler or applies it directly. Error handling and conflict resolution also function the same as they would in an Oracle-only environment. So, you can specify a prebuilt update conflict handler or create a custom conflict handler to resolve conflicts.

See Also:

- [Chapter 4, "Streams Apply Process"](#)
- [Chapter 5, "Rules"](#)
- [Chapter 7, "Streams Conflict Resolution"](#)
- ["Rule-Based Transformations"](#) on page 6-25

Instantiation from a Non-Oracle Database to an Oracle Database

There is no automatic way to instantiate tables that exist at a non-Oracle database at an Oracle database. However, you can perform the following general procedure to instantiate a table manually:

1. At the non-Oracle database, use a non-Oracle utility to export the table to a flat file.
2. At the Oracle database, create an empty table that matches the table at the non-Oracle database.
3. At the Oracle database, use `SQL*Loader` to load the contents of the flat file into the table.

See Also: *Oracle9i Database Utilities* for information about using `SQL*Loader`

Non-Oracle to Non-Oracle Data Sharing with Streams

Streams supports data sharing between two non-Oracle databases through a combination of non-Oracle to Oracle data sharing and Oracle to non-Oracle data sharing. Such an environment would use Streams in an Oracle database as an intermediate database between two non-Oracle databases.

For example, a non-Oracle to non-Oracle environment may consist of the following databases:

- A non-Oracle database named `het1.net`
- An Oracle database named `dbs1.net`
- A non-Oracle database named `het2.net`

A user application assembles changes at `het1.net` and enqueues them into a queue in `dbs1.net`. Then, the apply process at `dbs1.net` applies the changes to `het2.net` using Heterogeneous Services and a gateway. Another apply process at `dbs1.net` could apply some or all of the changes in the queue locally at `dbs1.net`. One or more propagations at `dbs1.net` could propagate some or all of the changes in the queue to other Oracle databases.

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](#)
- [Protection from Failures](#)
- [Best Practices for Streams High Availability Environments](#)

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. Data Guard, in SQL apply mode, implements a logical standby database in a high availability environment. Because 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 cluster, and failover and switchover between replicas.

See Also:

- *Oracle9i Data Guard Concepts and Administration* for more information about Data Guard
- *Oracle9i Real Application Clusters Concepts*

Protection from Failures

Oracle Real Application Clusters 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 standby 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 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:

- *Oracle9i Backup and Recovery Concepts* for more information about database backups and mirroring or multiplexing database files
- *Oracle9i 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 locally capture changes 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. However, there is the possibility for a small window of data loss in the event of a catastrophic failure, which can be avoided by the use of a companion physical standby database.

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.

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 need not be running on the exact same platform. This gives 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 Data Guard use the `LOG_ARCHIVE_DEST_n` initialization parameter to specify destinations, there is a limit of ten copies when you use Data Guard.

Fast Failover

Streams replicas can be open to read/write operations at all times. If a primary database fails, 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 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 lost.

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.

Coexistence with Streams

If you wish to have a logical standby database, but are also using Streams on the primary database, you must use Streams APIs to create and maintain the logical copy of the database. Data Guard SQL apply mode and Streams are not supported on the same database.

When Not to Use Streams

As mentioned previously, there are scenarios where customers may choose to use Streams to meet some of their 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. Customers who decide to leverage the flexibility offered by Streams 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 out-of-the-box with Oracle Data Guard.

Also, because Streams was designed for data integration, not high availability, it does not provide a zero data loss mode of operation, as does Oracle Data Guard. Customers who cannot afford to lose transactions in the event of a failure should either use Data Guard rather than Streams, or complement their Streams-based solution with a zero data loss physical standby maintained by Data Guard. Data Guard also provides a delayed apply option to protect from human errors. Again, a complimentary physical standby can provide such protection.

Application Maintained Copies

The best availability can be achieved by designing the maintenance of logical copies of data directly into an application. The application knows what data is valuable and must be immediately moved off-site to guarantee no data loss. It can also synchronously replicate truly critical data, while asynchronously replicating less critical data. Applications maintain copies of data by either synchronously or asynchronously sending data to other applications that manage another logical copy of the data. Synchronous operations are performed using the distributed SQL or remote procedure features of the database. Asynchronous operations are performed using Advanced Queuing. Advanced Queuing is a database integrated message queuing feature built on top of the infrastructure 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 Data Guard and Streams replication must be re-analyzed and solved by the custom application developers. In addition, standard solutions like 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: *Oracle9i Application Developer's Guide - Advanced Queuing* 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**
- **Recovering from Failures**
 - **Reestablishing Database Links After a Failover**
 - **Restarting Capture After a Failover**
 - **Restarting Propagation After a Failover**
 - **Restarting Apply 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:

- ["Each Databases Is a Source and Destination Database for Shared Data"](#) on page 8-8 and [Chapter 23, "Multiple Source Replication Example"](#) 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 unmanageable 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 from failures also apply to distributed hub databases. Oracle Corporation recommends Oracle Real Application Clusters 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: ["Primary Database Sharing Data with Several Secondary Databases"](#) on page 8-12 for a detailed example of such an environment

Configuring Oracle Real Application Clusters with Streams

Using Oracle Real Application Clusters with Streams introduces some important considerations. Capturing changes from the online redo log as it is written is not supported with Oracle Real Application Clusters. Rather, changes are captured from the archived redo log. Capturing from the archived redo logs introduces additional latency between the time a change is made at the production database and the time it appears at the replica.

If low latency is important, a cold failover cluster should be used to protect from system failure rather than Oracle Real Application Clusters. 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.

When running in an Oracle Real Application Clusters 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 where propagation is enabled. Assuming propagation is enabled for an instance, a propagation job running on that 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 Real Application Clusters 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.

See Also:

- ["Streams Capture Processes and Oracle Real Application Clusters"](#) on page 2-17
- ["Streams Queues and Oracle Real Application Clusters"](#) on page 3-18
- ["Streams Apply Processes and Oracle Real Application Clusters"](#) on page 4-29

Recovering from Failures

The following sections provide best practices for recovering from failures.

Reestablishing Database Links After a Failover

It is important to ensure that propagation continues to function after a failure of a destination database instance. A propagation job will retry (with increasing delay between retries) the database link continually after a failure until the connection is reestablished. In the event the database is restarted on the same node, or on a different node in a cold failover cluster, the connection should be reestablished once the database instance is restarted. In some circumstances, the database link may be waiting on a read or write, and will not detect the failure until a lengthy timeout, controlled by the `TCP_KEEPA_LIVE_INTERVAL` TCP/IP parameter, expires. In such

circumstances, the database link should be dropped and re-created to ensure that communication is reestablished quickly.

When an instance in an Oracle Real Application Clusters 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. Any inbound database links must be dropped and reestablished to point to the new instance that owns the destination queue. In a high availability environment, you can prepare scripts that will drop and re-create all necessary database links. Once a failed instance has been recovered, you can execute these scripts so that Streams can resume propagation.

See Also: ["Configuring Network Connectivity and Database Links"](#) on page 11-14 for information about creating database links in a Streams environment

Restarting Capture 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 state it was in at the time of the failure. That is, if it was running at the time of the failure, there is no need to restart the capture process.

For a capture process running in an Oracle Real Application Clusters 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. You must determine which instance now owns the queue used by the capture process by querying the `DBA_QUEUE_TABLES` data dictionary view, and then restart the capture process on that node. If the failed instance is brought back online subsequently, it will not restart the capture process, even though it was running the capture process at the time of failure, because it is no longer the owner of the queue used by the capture process.

See Also:

- ["Streams Capture Processes and Oracle Real Application Clusters"](#) on page 2-17
- ["Starting a Capture Process"](#) on page 12-5

Restarting Propagation 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 a Real Application Cluster environment, care must be taken to ensure the propagation jobs are enabled on the correct instance. A propagation job should run on the instance that owns the source queues from which the propagation job sends events to destination queues. An instance affinity parameter can be specified for the propagation job that will force the job to run on a particular instance. If the instance fails, and the queue ownership migrates to another instance, then the propagation job affinity must be reset to this other instance. Also, for any jobs to run on an instance, the dynamic initialization parameter `JOB_QUEUE_PROCESSES` must be greater than zero for that instance.

See Also: ["Streams Queues and Oracle Real Application Clusters"](#) on page 3-18

Restarting Apply 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 state it was in at the time of the failure. That is, if it was running at the time of the failure, then there is no need to restart the apply process.

In an Oracle Real Application Clusters cluster, if an instance hosting the apply process fails, the queue from which the apply process dequeues the events will be assigned to another node in the cluster. You need to determine which instance now owns the queue used by the apply process by querying the `DBA_QUEUE_TABLES` data dictionary view, and then restart the apply process on that node. If the failed instance is brought back online subsequently, it will not restart the apply process, even though it was running the apply process at the time of failure, because it is no longer the owner of the queue used by the apply process.

See Also:

- ["Streams Apply Processes and Oracle Real Application Clusters"](#) on page 4-29
- ["Starting an Apply Process"](#) on page 14-7

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:

- [Chapter 11, "Configuring a Streams Environment"](#)
- [Chapter 12, "Managing a Capture Process"](#)
- [Chapter 13, "Managing Staging and Propagation"](#)
- [Chapter 14, "Managing an Apply Process"](#)
- [Chapter 15, "Managing Rules and Rule-Based Transformations"](#)
- [Chapter 16, "Other Streams Management Tasks"](#)
- [Chapter 17, "Monitoring a Streams Environment"](#)
- [Chapter 18, "Troubleshooting a Streams Environment"](#)

Configuring 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](#)
- [Setting Initialization Parameters Relevant to Streams](#)
- [Setting Export and Import Parameters Relevant to Streams](#)
- [Configuring a Database to Run a Streams Capture Process](#)
- [Configuring Network Connectivity and Database Links](#)
- [Configuring a Capture-Based Streams Environment](#)

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 as an administrative user who can create users, grant privileges, create tablespaces, and alter users.
2. Create a new user to act as the Streams administrator or use an existing user. For example, to create a new user named `strmadmin`, run the following statement:

```
CREATE USER strmadmin IDENTIFIED BY strmadminpw;
```

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

3. Grant the Streams administrator at least the following privileges:

```
GRANT CONNECT, RESOURCE TO strmadmin;
GRANT EXECUTE ON DBMS_AQADM TO strmadmin;
GRANT EXECUTE ON DBMS_STREAMS_ADM TO strmadmin;

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

BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege => DBMS_RULE_ADM.CREATE_RULE_OBJ,
    grantee   => 'strmadmin',
    grant_option => FALSE);
END;
/
```

4. If necessary, grant the Streams administrator the following privileges:
 - EXECUTE privilege on the DBMS_APPLY_ADM package if the Streams administrator will manage one or more apply processes on the database. The Streams administrator also must have EXECUTE privilege on any apply handlers and error handlers configured using the subprograms in the DBMS_APPLY_ADM package.
 - EXECUTE privilege on the DBMS_CAPTURE_ADM package if the Streams administrator will manage one or more capture processes on the database
 - EXECUTE privilege on the DBMS_PROPAGATION_ADM package if the Streams administrator will manage one or more propagations on the database
 - EXECUTE privilege on the DBMS_FLASHBACK package if the Streams administrator will need to obtain the current SCN for a database. Typically, the Streams administrator must determine the current SCN to set an instantiation SCN using the SET_TABLE_INSTANTIATION_SCN, SET_SCHEMA_INSTANTIATION_SCN, or SET_GLOBAL_INSTANTIATION_SCN procedure in the DBMS_APPLY_ADM package.
 - SELECT_CATALOG_ROLE if you want to enable the Streams administrator to monitor the environment easily
 - SELECT ANY DICTIONARY privilege if you plan to use the Streams tool in Oracle Enterprise Manager
 - SELECT privilege on the DBA_APPLY_ERROR data dictionary view if you want the Streams administrator to be able to select from this view within a PL/SQL subprogram. See ["Displaying Detailed Information About Apply Errors"](#) on page 17-37 for an example of such a PL/SQL subprogram.
 - 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, or apply process. For an apply process, if an apply user is specified, then the apply user must have these privileges.
 - If the Streams administrator does not own the queue used by a Streams capture process, propagation, or apply process, 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 13-3 for instructions.
5. 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 25 M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;
```
 6. Specify the tablespace for the Streams administrator:

```
ALTER USER strmadmin DEFAULT TABLESPACE streams_tbs  
    QUOTA UNLIMITED ON streams_tbs;
```
 7. Repeat all of the previous steps at each database in the environment that will use Streams.

Setting Initialization Parameters Relevant to Streams

[Table 11-1](#) lists initialization parameters that are important for the operation, reliability, and performance of a Streams environment. Set these parameters appropriately for your Streams environment.

See Also: *Oracle9i Database Reference* for more information about these initialization parameters

Table 11–1 Initialization Parameters Relevant to Streams (Page 1 of 4)

Parameter	Values	Description
AQ_TM_PROCESSES	Default: 0 Range: 0 to 10	Establishes queue monitor processes. Setting the parameter to 1 or more starts the specified number of queue monitor processes. These queue monitor processes are responsible for managing time-based operations of messages such as delay and expiration, cleaning up retained messages after the specified retention time, and cleaning up consumed messages if the retention time is zero. If you want to enqueue user events into a Streams queue, then set this parameter to 1 or higher. User events are those created by users and applications, not by a Streams capture process.
ARCHIVE_LAG_TARGET	Default: 0 Range: 0 or any integer in [60, 7200]	Limits the amount of data that can be lost and effectively increases the availability of the standby database by forcing a log switch after a user-specified time period elapses. If you are using Streams in a Real Application Clusters environment, then set this parameter to a value greater than zero to switch the log files automatically. See Also: "Streams Capture Processes and Oracle Real Application Clusters" on page 2-17
COMPATIBLE	Default: 8.1.0 Range: 8.1.0 to Current Release Number	This parameter specifies the release with which the Oracle server must maintain compatibility. Oracle servers with different compatibility levels can interoperate. To use Streams, this parameter must be set to 9.2.0 or higher.
GLOBAL_NAMES	Default: false Range: true or false	Specifies whether a database link is required to have the same name as the database to which it connects. If you want to use Streams to share information between databases, then set this parameter to true at each database that is participating in your Streams environment.

Table 11–1 Initialization Parameters Relevant to Streams (Page 2 of 4)

Parameter	Values	Description
JOB_QUEUE_PROCESSES	Default: 0 Range: 0 to 1000	Specifies the number of <i>Jn</i> job queue processes for each instance (J000 ... J999). Job queue processes handle requests created by DBMS_JOB. You can change the setting for JOB_QUEUE_PROCESSES dynamically by using the ALTER SYSTEM statement. 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.
LOG_PARALLELISM	Default: 1 Range: 1 to 255	Specifies the level of concurrency for redo allocation within Oracle. If you plan to run one or more capture processes on a database, then this parameter must be set to 1. Setting this parameter to 1 does not affect the parallelism of capture. You can set parallelism for a capture process using the SET_PARAMETER procedure in the DBMS_CAPTURE_ADM package.
LOGMNR_MAX_PERSISTENT_SESSIONS	Default: 1 Range: 1 to LICENSE_MAX_SESSIONS	Specifies the maximum number of persistent LogMiner mining sessions that are concurrently active when all sessions are mining redo logs generated by instances. If you plan to run multiple Streams capture processes on a single database, then set this parameter equal to or higher than the number of planned capture processes.
OPEN_LINKS	Default: 4 Range: 0 to 255	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.

Table 11–1 Initialization Parameters Relevant to Streams (Page 3 of 4)

Parameter	Values	Description
PARALLEL_MAX_SERVERS	<p>Default: Derived from the values of the following parameters:</p> <p>CPU_COUNT</p> <p>PARALLEL_ADAPTIVE_MULTI_USER</p> <p>PARALLEL_AUTOMATIC_TUNING</p> <p>Range: 0 to 3599</p>	<p>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.</p> <p>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.</p>
PROCESSES	<p>Default: Derived from PARALLEL_MAX_SERVERS</p> <p>Range: 6 to operating system dependent limit</p>	<p>Specifies the maximum number of operating system user processes that can simultaneously connect to Oracle.</p> <p>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.</p>
SESSIONS	<p>Default: Derived from:</p> <p>$(1.1 * PROCESSES) + 5$</p> <p>Range: 1 to 2^{31}</p>	<p>Specifies the maximum number of sessions that can be created in the system.</p> <p>If you plan to run one or more capture processes or apply processes in a database, then you may need to increase the size of this parameter. Each background process in a database requires a session.</p>
SGA_MAX_SIZE	<p>Default: Initial size of SGA at startup</p> <p>Range: 0 to operating system dependent limit</p>	<p>Specifies the maximum size of SGA for the lifetime of a database instance.</p> <p>If you plan to run multiple capture processes on a single database, then you may need to increase the size of this parameter.</p>

Table 11–1 Initialization Parameters Relevant to Streams (Page 4 of 4)

Parameter	Values	Description
SHARED_POOL_SIZE	<p>Default:</p> <p>32-bit platforms: 8 MB, rounded up to the nearest granule size</p> <p>64-bit platforms: 64 MB, rounded up to the nearest granule size</p> <p>Range:</p> <p>Minimum: the granule size</p> <p>Maximum: operating system-dependent</p>	<p>Specifies (in bytes) the size of the shared pool. The shared pool contains shared cursors, stored procedures, control structures, and other structures.</p> <p>You should increase the size of the shared pool by 10 MB for each capture process on a database.</p>
TIMED_STATISTICS	<p>Default:</p> <p>If STATISTICS_LEVEL is set to TYPICAL or ALL, then true</p> <p>If STATISTICS_LEVEL is set to BASIC, then false</p> <p>The default for STATISTICS_LEVEL is TYPICAL.</p> <p>Range: true or false</p>	<p>Specifies whether or not statistics related to time are collected.</p> <p>If you want to collect elapsed time statistics in the data dictionary views related to Streams, then 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.</p>

Setting Export and Import Parameters Relevant to Streams

This section describes Export and Import utility parameters that are relevant to Streams.

See Also:

- ["Instantiation"](#) on page 2-13
- ["Performing Full Database Export/Import on a Database Using Streams"](#) on page 16-35
- *Oracle9i Database Utilities* for information about performing a exports and imports

Export Utility Parameters Relevant to Streams

The following Export utility parameter is relevant to Streams.

The `OBJECT_CONSISTENT` Export Utility Parameter and Streams

The `OBJECT_CONSISTENT` Export utility parameter specifies whether or not the Export utility repeatedly uses the `SET TRANSACTION READ ONLY` statement to ensure that the exported data and the exported procedural actions for each object are consistent to a single point in time. If `OBJECT_CONSISTENT` is set to `y`, then each object is exported in its own read-only transaction, even if it is partitioned. In contrast, if you use the `CONSISTENT` Export utility parameter, then there is only one read-only transaction.

When you perform an instantiation in a Streams environment, some degree of consistency is required in the export dump file. The `OBJECT_CONSISTENT` Export utility parameter is sufficient to ensure this consistency for Streams instantiations. If you are using an export dump file for other purposes in addition to a Streams instantiation, and these other purposes have more stringent consistency requirements than that provided by `OBJECT_CONSISTENT`, then you can use Export utility parameters `CONSISTENT`, `FLASHBACK_SCN`, or `FLASHBACK_TIME` for Streams instantiations.

By default the `OBJECT_CONSISTENT` Export utility parameter is set to `n`. Specify `y` when an export is performed as part of a Streams instantiation and no more stringent Export utility parameter is needed.

Attention:

- During an export for a Streams instantiation, make sure no DDL changes are made to objects being exported.
 - When you export a database or schema that contains rules with non-NULL action contexts, then the database or the default tablespace of the schema that owns the rules must be writeable. If the database or tablespace is read-only, then export errors result.
-
-

Import Utility Parameters Relevant to Streams

The following Import utility parameters are relevant to Streams.

The `STREAMS_INSTANTIATION` Import Utility Parameter and Streams

The `STREAMS_INSTANTIATION` Import utility parameter specifies whether to import Streams instantiation metadata that may be present in the export dump file. When this parameter is set to `y`, the import session sets its Streams tag to the hexadecimal equivalent of '00' to avoid cycling the changes made by the import. Redo entries resulting from the import have this tag value. By default the `STREAMS_INSTANTIATION` Import utility parameter is set to `n`. Specify `y` when an import is performed as part of a Streams instantiation.

See Also: [Chapter 8, "Streams Tags"](#)

The `STREAMS_CONFIGURATION` Import Utility Parameter and Streams

The `STREAMS_CONFIGURATION` Import utility parameter specifies whether to import any general Streams metadata that may be present in the export dump file. This import parameter is relevant only if you are performing a full database import. By default the `STREAMS_CONFIGURATION` Import utility parameter is set to `y`. Typically, specify `y` if an import is part of a backup or restore operation.

The following objects are imported regardless of the `STREAMS_CONFIGURATION` setting:

- Streams queues and their queue tables (if `STREAMS_CONFIGURATION` is set to `n`, then these queues are not started when they are imported)
- Queue subscribers
- Advanced queuing agents
- Job queue processes related to Streams propagations
- Rules, including their rule sets and evaluation contexts. All rules are imported, including Streams rules and non-Streams rules. Streams rules are rules generated by the system when certain procedures in the `DBMS_STREAMS_ADM` package are run, while non-Streams rules are rules created using the `DBMS_RULE_ADM` package.

If the `STREAMS_CONFIGURATION` parameter is set to `n`, then information about Streams rules is not imported into the following data dictionary views:

`ALL_STREAMS_GLOBAL_RULES`, `ALL_STREAMS_SCHEMA_RULES`,
`ALL_STREAMS_TABLE_RULES`, `DBA_STREAMS_GLOBAL_RULES`,

DBA_STREAMS_SCHEMA_RULES, and DBA_STREAMS_TABLE_RULES.

However, regardless of the STREAMS_CONFIGURATION parameter setting, information about these rules is imported into the ALL_RULES, ALL_RULE_SETS, ALL_RULE_SET_RULES, DBA_RULES, DBA_RULE_SETS, DBA_RULE_SET_RULES, USER_RULES, USER_RULE_SETS, and USER_RULE_SET_RULES data dictionary views.

When the STREAMS_CONFIGURATION Import utility parameter is set to `y`, the import includes the following information; when the STREAMS_CONFIGURATION Import utility parameter is set to `n`, the import does not include the following information:

- Capture processes, including the following information for each capture process:
 - Name of the capture process
 - State of the capture process
 - Capture process parameter settings
 - Queue owner and queue name of the queue used by the capture process
 - Rule set owner and rule set name of the rule set used by the capture process
- If any tables have been prepared for instantiation at the export database, then these tables are prepared for instantiation at the import database.
- If any schemas have been prepared for instantiation at the export database, then these schemas are prepared for instantiation at the import database.
- If the export database has been prepared for instantiation, then the import database is prepared for instantiation.
- The state of each Streams queue, either started or stopped (Streams queues themselves are imported regardless of the parameter setting)
- Propagations, including the following information for each propagation:
 - Name of the propagation
 - Queue owner and queue name of the source queue
 - Queue owner and queue name of the destination queue
 - Destination database link
 - Rule set owner and rule set name of the rule set used by the propagation

- Apply processes, including the following information for each apply process:
 - Name of the apply process
 - State of the apply process
 - Apply process parameter settings
 - Queue owner and queue name of the queue used by the apply process
 - Rule set owner and rule set name of the rule set used by the apply process
 - Whether the apply process applies captured or user-enqueued events
 - Apply user for the apply process, if one exists
 - Message handler used by the apply process, if one exists
 - DDL handler used by the apply process, if one exists
 - Tag generated in the redo log for changes made by the apply process
 - Apply database link, if one exists
 - Source database for the apply process
 - The information about apply progress in the `DBA_APPLY_PROGRESS` data dictionary view, including applied message number, oldest message number, apply time, and applied message create time
 - Apply errors
- DML handlers
- Error handlers
- Update conflict handlers
- Substitute key columns for apply tables
- Instantiation SCN for each apply object
- Ignore SCN for each apply object
- Some data dictionary information about Streams rules. The rules themselves are imported regardless of the setting for the `STREAMS_CONFIGURATION` parameter.

Configuring a Database to Run a Streams Capture Process

The following sections describe database requirements for running a Streams capture process:

- [Configuring the Database to Run in ARCHIVELOG Mode](#)
- [Specifying an Alternate Tablespace for LogMiner](#)

In addition to these tasks, make sure the initialization parameters are set properly on any database that will run a capture process.

See Also: ["Setting Initialization Parameters Relevant to Streams"](#) on page 11-4

Configuring the Database to Run in ARCHIVELOG Mode

Any database where changes are captured by a capture process must be running in ARCHIVELOG mode.

See Also:

- ["ARCHIVELOG Mode and a Capture Process"](#) on page 2-25
- *Oracle9i Database Administrator's Guide* for information about running a database in ARCHIVELOG mode

Specifying an Alternate Tablespace for LogMiner

By default, the LogMiner tables are in the SYSTEM tablespace, but the SYSTEM tablespace may not have enough space for these tables once a capture process starts to capture changes. Therefore, you must create an alternate tablespace for the LogMiner tables.

The following example creates a tablespace named `logmnrts` for use by LogMiner:

1. Connect as an administrative user who has privileges to create tablespaces and execute subprograms in the `DBMS_LOGMNR_D` package.
2. Either create an alternate tablespace for the LogMiner tables or use an existing tablespace. For example, the following statement creates an alternate tablespace for the LogMiner tables:

```
CREATE TABLESPACE logmnrts DATAFILE '/usr/oracle/dbs/logmnrts.dbf'  
    SIZE 25 M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;
```

3. Run the `SET_TABLESPACE` procedure in the `DBMS_LOGMNR_D` package to set the alternate tablespace for LogMiner. For example, to specify a tablespace named `logmnrts`, run the following procedure:

```
EXECUTE DBMS_LOGMNR_D.SET_TABLESPACE('logmnrts');
```

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: *Oracle9i 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: *Oracle9i 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 `db2.net` connecting as a Streams administrator named `strmadmin`, run the following statement:

```
CREATE DATABASE LINK db2.net CONNECT TO strmadmin IDENTIFIED BY strmadminpw  
USING 'db2.net';
```

See Also: *Oracle9i Database Administrator's Guide* for more information about creating database links

Configuring a Capture-Based Streams Environment

This section describes the general steps for performing the following tasks:

- [Creating a New Streams Single Source Environment](#)
- [Adding Shared Objects to an Existing Single Source Environment](#)
- [Adding a New Destination Database to an Existing Single Source Environment](#)
- [Creating a New Multiple Source Environment](#)
- [Adding Shared Objects to an Existing Multiple Source Environment](#)
- [Adding a New Database to an Existing Multiple Source Environment](#)

Note: The instructions in the following sections assume you will use the `DBMS_STREAMS_ADM` package to configure your Streams environment. If you use other packages, then extra steps may be necessary for each task.

See Also: The following chapters for detailed examples of configuring Streams capture-based environments:

- [Chapter 20, "Single Database Capture and Apply Example"](#)
- [Chapter 21, "Simple Single Source Replication Example"](#)
- [Chapter 22, "Single Source Heterogeneous Replication Example"](#)
- [Chapter 23, "Multiple Source Replication Example"](#)

Creating a New Streams Single Source Environment

This section lists the general steps to perform when creating a new single source Streams environment. A single source environment is one in which there is only one source database for shared data. There may be more than one source database in a single source environment, but in this case no two source databases capture any of the same data.

Before starting capture processes and configuring propagations in a new Streams environment, make sure any propagations or apply processes that will receive events are configured to handle these events. That is, the propagations or apply processes should exist, and each one should be associated with a rule set that

handles the events appropriately. If these propagations and apply processes are not configured properly to handle these events, then events may be lost.

In general, if you are configuring a new Streams environment in which changes for shared objects are captured at one database and then propagated and applied at remote databases, then you should configure the environment in the following order:

1. Complete the necessary tasks described previously in this chapter to prepare each database in your environment for Streams:
 - ["Configuring a Streams Administrator"](#) on page 11-2
 - ["Setting Initialization Parameters Relevant to Streams"](#) on page 11-4
 - ["Configuring a Database to Run a Streams Capture Process"](#) on page 11-13
 - ["Configuring Network Connectivity and Database Links"](#) on page 11-14

Some of these tasks may not be required at certain databases.

2. Create any necessary Streams queues that do not already exist. When you create a capture process or apply process, you associate the process with a specific Streams queue. When you create a propagation, you associate it with a specific source queue and destination queue. See ["Creating a Streams Queue"](#) on page 13-2 for instructions.
3. Specify supplemental logging at each source database for any shared object. See ["Specifying Supplemental Logging at a Source Database"](#) on page 12-9 for instructions.
4. At each database, create the required capture processes, propagations, and apply processes for your environment. You can create them in any order.
 - Create one or more capture processes at each database that will capture changes. Make sure each capture process uses a rule set that is appropriate for capturing changes. Do not start the capture processes you create. See ["Creating a Capture Process"](#) on page 12-2 for instructions.

When you use the `DBMS_STREAMS_ADM` package to add the capture rules, it automatically runs the `PREPARE_TABLE_INSTANTIATION`, `PREPARE_SCHEMA_INSTANTIATION`, or `PREPARE_GLOBAL_INSTANTIATION` procedure in the `DBMS_CAPTURE_ADM` package for the specified table, specified schema, or entire database, respectively.

You must run the appropriate procedure to prepare for instantiation manually if any of the following conditions is true:

- You use the `DBMS_RULE_ADM` package to add or modify capture rules.
- You use an existing capture process and do not add capture rules for any shared object.

If you must prepare for instantiation manually, then see "[Preparing Database Objects for Instantiation at a Source Database](#)" on page 12-11 for instructions.

- Create all propagations that propagate the captured events from a source queue to a destination queue. Make sure each propagation uses a rule set that is appropriate for propagating changes. See "[Creating a Propagation](#)" on page 13-8 for instructions.
 - Create one or more apply processes at each database that will apply changes. Make sure each apply process uses a rule set that is appropriate for applying changes. Do not start the apply processes you create. See "[Creating an Apply Process](#)" on page 14-2 for instructions.
5. Either instantiate, or set the instantiation SCN for, each database object for which changes are applied by an apply process. If a database object does not exist at a destination database, then instantiate it using Export/Import. If a database object already exists at a destination database, then set the instantiation SCN for it manually.
- To instantiate database objects using Export/Import, first export them at the source database with the `OBJECT_CONSISTENT` export parameter set to `y`, or use a more stringent degree of consistency. Then, import them at the destination database with the `STREAMS_INSTANTIATION` import parameter set to `y`. See "[Setting Instantiation SCNs Using Export/Import](#)" on page 14-36 for information.
 - To set the instantiation SCN for a table, schema, or database manually, run the appropriate procedure or procedures in the `DBMS_APPLY_ADM` package at the destination database:
 - `SET_TABLE_INSTANTIATION_SCN`
 - `SET_SCHEMA_INSTANTIATION_SCN`
 - `SET_GLOBAL_INSTANTIATION_SCN`

When you run one of these procedures, you must ensure that the shared objects at the destination database are consistent with the source database as of the instantiation SCN.

If you run `SET_GLOBAL_INSTANTIATION_SCN` at a destination database, then you must also run `SET_SCHEMA_INSTANTIATION_SCN` for each existing schema in the source database whose DDL changes you are applying, and you must run `SET_TABLE_INSTANTIATION_SCN` for each existing table in the source database whose DML or DDL changes you are applying.

If you run `SET_SCHEMA_INSTANTIATION_SCN` at a destination database, then you must also run `SET_TABLE_INSTANTIATION_SCN` for each existing source database table in the schema whose DML or DDL changes you are applying.

See "[Setting Instantiation SCNs Using the DBMS_APPLY_ADM Package](#)" on page 14-38 for instructions.

Alternatively, you can perform a metadata export/import to set the instantiation SCNs for existing database objects. If you choose this option, then make sure no rows are imported. Also, make sure the shared objects at all of the destination databases are consistent with the source database that performed the export at the time of the export. If you are sharing DML changes only, then table level export/import is sufficient. If you are sharing DDL changes also, then additional considerations apply. See "[Setting Instantiation SCNs Using Export/Import](#)" on page 14-36 for more information about performing a metadata export/import.

6. Start each apply process you created in Step 4. See "[Starting an Apply Process](#)" on page 14-7 for instructions.
7. Start each capture process you created in Step 4. See "[Starting a Capture Process](#)" on page 12-5 for instructions.

When you are configuring the environment, remember that capture processes and apply processes are stopped when they are created, but propagations are scheduled to propagate events immediately when they are created. The capture process must be created before the relevant objects are instantiated at a remote destination database. You must create the propagations and apply processes before starting the capture process, and you must instantiate the objects before running the whole stream.

See Also: [Chapter 21, "Simple Single Source Replication Example"](#) and [Chapter 22, "Single Source Heterogeneous Replication Example"](#) for detailed examples that set up single source environments

Adding Shared Objects to an Existing Single Source Environment

You add existing database objects to an existing single source environment by adding the necessary rules to the appropriate capture processes, propagations, and apply processes. Before creating or altering capture or propagation rules in a running Streams environment, make sure any propagations or apply processes that will receive events as a result of the new or altered rules are configured to handle these events. That is, the propagations or apply processes should exist, and each one should be associated with a rule set that handles the events appropriately. If these propagations and apply processes are not configured properly to handle these events, then events may be lost.

For example, suppose you want to add a table to a Streams environment that already captures, propagates, and applies changes to other tables. Assume only one capture process will capture changes to this table, and only one apply process will apply changes to this table. In this case, you must add one or more table-level rules to the following rule sets:

- The rule set for the capture process that will capture changes to the table
- The rule set for each propagation that will propagate changes to the table
- The rule set for the apply process that will apply changes to the table

If you perform administrative steps in the wrong order, you may lose events. For example, if you add the rule to the capture rule set first, without stopping the capture process, then the propagation will not propagate the changes if it does not have a rule that instructs it to do so, and the changes may be lost.

To avoid losing events, you should complete the configuration in the following order:

1. Either stop the capture process, disable one of the propagation jobs, or stop the apply processes. See one of the following sections for instructions:
 - ["Stopping a Capture Process"](#) on page 12-14
 - ["Disabling a Propagation Job"](#) on page 13-17
 - ["Stopping an Apply Process"](#) on page 14-7

2. Add the relevant rules to the rule sets for the propagations and the apply processes. See the following sections for instructions:
 - ["Adding Rules to the Rule Set for a Propagation"](#) on page 13-14
 - ["Adding Rules to the Rule Set for an Apply Process"](#) on page 14-8
3. Add the relevant rules to the rule set used by the capture process. See ["Adding Rules to the Rule Set for a Capture Process"](#) on page 12-5 for instructions.

When you use the `DBMS_STREAMS_ADM` package to add the capture rules, it automatically runs the `PREPARE_TABLE_INSTANTIATION`, `PREPARE_SCHEMA_INSTANTIATION`, or `PREPARE_GLOBAL_INSTANTIATION` procedure in the `DBMS_CAPTURE_ADM` package for the specified table, specified schema, or entire database, respectively.

You must run the appropriate procedure to prepare for instantiation manually if any of the following conditions is true:

- You use `DBMS_RULE_ADM` to create or modify rules in the capture process rule set.
- You do not add rules for the added objects to the capture process rule set, because the capture process already captures changes to these objects. In this case, rules for the objects may be added to propagations and apply processes in the environment, but not to the capture process.

If you must prepare for instantiation manually, then see ["Preparing Database Objects for Instantiation at a Source Database"](#) on page 12-11 for instructions.

4. At each destination database, either instantiate, or set the instantiation SCN for, each database object you are adding to the Streams environment. If a database object does not exist at a destination database, then instantiate it using Export/Import. If a database object exists at a destination database, then set the instantiation SCN for it.
 - To instantiate database objects using Export/Import, first export them at the source database with the `OBJECT_CONSISTENT` export parameter set to `y`, or use a more stringent degree of consistency. Then, import them at the destination database with the `STREAMS_INSTANTIATION` import parameter set to `y`. See ["Setting Instantiation SCNs Using Export/Import"](#) on page 14-36 for information.

- To set the instantiation SCN for a table, schema, or database manually, run the appropriate procedure or procedures in the `DBMS_APPLY_ADM` package at a destination database:
 - `SET_TABLE_INSTANTIATION_SCN`
 - `SET_SCHEMA_INSTANTIATION_SCN`
 - `SET_GLOBAL_INSTANTIATION_SCN`

When you run one of these procedures at a destination database, you must ensure that every added object at the destination database is consistent with the source database as of the instantiation SCN.

If you run `SET_GLOBAL_INSTANTIATION_SCN` at a destination database, then you must also run `SET_SCHEMA_INSTANTIATION_SCN` for each existing source database schema whose changes you are applying and `SET_TABLE_INSTANTIATION_SCN` for each existing source database table whose changes you are applying.

If you run `SET_SCHEMA_INSTANTIATION_SCN` at a destination database, then you must also run `SET_TABLE_INSTANTIATION_SCN` for each existing source database table in the schema whose DML or DDL changes you are applying.

See ["Setting Instantiation SCNs Using the DBMS_APPLY_ADM Package"](#) on page 14-38 for instructions.

Alternatively, you can perform a metadata export/import to set the instantiation SCNs for existing database objects. If you choose this option, then make sure no rows are imported. Also, make sure every added object at the importing destination database is consistent with the source database that performed the export at the time of the export. If you are sharing DML changes only, then table level export/import is sufficient. If you are sharing DDL changes also, then additional considerations apply. See ["Setting Instantiation SCNs Using Export/Import"](#) on page 14-36 for more information about performing a metadata export/import.

5. Start any process you stopped in Step 1 or enable any propagation job you disabled in Step 1. See one of the following sections for instructions:
 - ["Starting a Capture Process"](#) on page 12-5
 - ["Enabling a Propagation Job"](#) on page 13-11
 - ["Starting an Apply Process"](#) on page 14-7

You must stop the capture process, disable one of the propagation jobs, or stop the apply process in Step 1 to ensure that the table or schema is instantiated before the first LCR resulting from the added rule(s) reaches the apply process. Otherwise, events could be lost or could result in apply errors, depending on whether the apply rule(s) have been added.

If you are certain that the added table is not being modified at the source database during this procedure, and that there are no LCRs for the table already in the stream or waiting to be captured, then you can perform Step 5 before Step 4 to reduce the amount of time that a process or propagation job is stopped.

See Also: ["Add Objects to an Existing Streams Replication Environment"](#) on page 22-64 for a detailed example that adds objects to an existing single source environment

Adding a New Destination Database to an Existing Single Source Environment

You add a destination database to an existing single source environment by creating one or more new apply processes at the new destination database and, if necessary, configuring one or more propagations to propagate changes to the new destination database. You may also need to add rules to existing propagations in the stream that propagates to the new destination database.

As in the example that describes ["Adding Shared Objects to an Existing Single Source Environment"](#) on page 11-19, before creating or altering propagation rules in a running Streams environment, make sure any propagations or apply processes that will receive events as a result of the new or altered rules are configured to handle these events. Otherwise, events may be lost.

To avoid losing events, you should complete the configuration in the following order:

1. Complete the necessary tasks described previously in this chapter to prepare the new destination database for Streams:
 - ["Configuring a Streams Administrator"](#) on page 11-2
 - ["Setting Initialization Parameters Relevant to Streams"](#) on page 11-4
 - ["Configuring Network Connectivity and Database Links"](#) on page 11-14

Some of these tasks may not be required at the new database.

2. Create any necessary Streams queues that do not already exist at the destination database. When you create an apply process, you associate the apply process with a specific Streams queue. See ["Creating a Streams Queue"](#) on page 13-2 for instructions.
3. Create one or more apply processes at the new destination database to apply the changes from its source databases. Make sure each apply process uses a rule set that is appropriate for applying changes. Do not start any of the apply processes at the new database. See ["Creating an Apply Process"](#) on page 14-2 for instructions.

Keeping the apply processes stopped prevents changes made at the source databases from being applied before the instantiation of the new database is completed, which would otherwise lead to incorrect data and errors.

4. Configure any necessary propagations to propagate changes from the source databases to the new destination database. Make sure each propagation uses a rule set that is appropriate for propagating changes. See ["Creating a Propagation"](#) on page 13-8.
5. At the source database, prepare for instantiation each database object for which changes will be applied by an apply process at the new destination database. Run either the `PREPARE_TABLE_INSTANTIATION`, `PREPARE_SCHEMA_INSTANTIATION`, or `PREPARE_GLOBAL_INSTANTIATION` procedure in the `DBMS_CAPTURE_ADM` package for the specified table, specified schema, or entire database, respectively. See ["Preparing Database Objects for Instantiation at a Source Database"](#) on page 12-11 for instructions.
6. At the new destination database, either instantiate, or set the instantiation SCNs for, each database object for which changes will be applied by an apply process. If the database objects do not already exist at the new destination database, then instantiate them using Export/Import. If the database objects exist at the new destination database, then set the instantiation SCN for them.
 - To instantiate database objects using Export/Import, first export them at the source database with the `OBJECT_CONSISTENT` export parameter set to `y`, or use a more stringent degree of consistency. Then, import them at the new destination database with the `STREAMS_INSTANTIATION` import parameter set to `y`. See ["Setting Instantiation SCNs Using Export/Import"](#) on page 14-36 for information.

- To set the instantiation SCN for a table, schema, or database manually, run the appropriate procedure or procedures in the `DBMS_APPLY_ADM` package at the new destination database:

- `SET_TABLE_INSTANTIATION_SCN`
- `SET_SCHEMA_INSTANTIATION_SCN`
- `SET_GLOBAL_INSTANTIATION_SCN`

When you run one of these procedures, you must ensure that the shared objects at the new destination database are consistent with the source database as of the instantiation SCN.

If you run `SET_GLOBAL_INSTANTIATION_SCN` at a destination database, then you must also run `SET_SCHEMA_INSTANTIATION_SCN` for each existing schema in the source database whose DDL changes you are applying, and you must run `SET_TABLE_INSTANTIATION_SCN` for each existing table in the source database whose DML or DDL changes you are applying.

If you run `SET_SCHEMA_INSTANTIATION_SCN` at a destination database, then you must also run `SET_TABLE_INSTANTIATION_SCN` for each existing source database table in the schema whose DML or DDL changes you are applying.

See "[Setting Instantiation SCNs Using the DBMS_APPLY_ADM Package](#)" on page 14-38 for instructions.

Alternatively, you can perform a metadata export/import to set the instantiation SCNs for existing database objects. If you choose this option, then make sure no rows are imported. Also, make sure the shared objects at the importing destination database are consistent with the source database that performed the export at the time of the export. If you are sharing DML changes only, then table level export/import is sufficient. If you are sharing DDL changes also, then additional considerations apply. See "[Setting Instantiation SCNs Using Export/Import](#)" on page 14-36 for more information about performing a metadata export/import.

7. Start the apply processes you created in Step 3. See "[Starting an Apply Process](#)" on page 14-7 for instructions.

See Also: "[Add a Database to an Existing Streams Replication Environment](#)" on page 22-76 for detailed example that adds a database to an existing single source environment

Creating a New Multiple Source Environment

This section lists the general steps to perform when creating a new multiple source Streams environment. A multiple source environment is one in which there is more than one source database for any of the shared data.

This example uses the following terms:

- **Populated database:** A database that already contains the shared database objects before you create the new multiple source environment. You must have at least one populated database to create the new Streams environment.
- **Export database:** A populated database on which you perform an export of the shared database objects. This export is used to instantiate the shared database objects at the import databases. You may not have an export database if all of the databases in the environment are populated databases.
- **Import database:** A database that does not contain the shared database objects before you create the new multiple source environment. You instantiate the shared database objects at an import database using the export dump file from the export database. You may not have any import databases if all of the databases in the environment are populated databases.

Complete the following steps to create a new multiple source environment:

Note: Make sure no changes are made to the objects being shared at a database you are adding to the Streams environment until the instantiation at the database is complete.

1. Complete the necessary tasks described previously in this chapter to prepare each database in the environment for Streams:
 - ["Configuring a Streams Administrator"](#) on page 11-2
 - ["Setting Initialization Parameters Relevant to Streams"](#) on page 11-4
 - ["Configuring a Database to Run a Streams Capture Process"](#) on page 11-13
 - ["Configuring Network Connectivity and Database Links"](#) on page 11-14

Some of these tasks may not be required at certain databases.

2. At each populated database, specify any necessary supplemental logging for the shared objects. See ["Specifying Supplemental Logging at a Source Database"](#) on page 12-9 for instructions.

3. Create any necessary Streams queues that do not already exist. When you create a capture process or apply process, you associate the process with a specific Streams queue. When you create a propagation, you associate it with a specific source queue and destination queue. See ["Creating a Streams Queue"](#) on page 13-2 for instructions.
4. At each database, create the required capture processes, propagations, and apply processes for your environment. You can create them in any order.

- Create one or more capture processes at each database that will capture changes. Make sure each capture process uses a rule set that is appropriate for capturing changes. Do not start the capture processes you create. See ["Creating a Capture Process"](#) on page 12-2 for instructions.

When you use the `DBMS_STREAMS_ADM` package to add the capture rules, it automatically runs the `PREPARE_TABLE_INSTANTIATION`, `PREPARE_SCHEMA_INSTANTIATION`, or `PREPARE_GLOBAL_INSTANTIATION` procedure in the `DBMS_CAPTURE_ADM` package for the specified table, specified schema, or entire database, respectively.

You must run the appropriate procedure to prepare for instantiation manually if any of the following conditions is true:

- You use the `DBMS_RULE_ADM` package to add or modify capture rules.
- You use an existing capture process and do not add capture rules for any shared object.

If you must prepare for instantiation manually, then see ["Preparing Database Objects for Instantiation at a Source Database"](#) on page 12-11 for instructions.

- Create all propagations that propagate the captured events from a source queue to a destination queue. Make sure each propagation uses a rule set that is appropriate for propagating changes. See ["Creating a Propagation"](#) on page 13-8 for instructions.
- Create one or more apply processes at each database that will apply changes. Make sure each apply process uses a rule set that is appropriate for applying changes. Do not start the apply processes you create. See ["Creating an Apply Process"](#) on page 14-2 for instructions.

After completing these steps, complete the steps in each of the following sections that apply to your environment. You may need to complete the steps in only one of these sections or in both of these sections:

- For each populated database, complete the steps in "[Configuring Populated Databases When Creating a Multiple Source Environment](#)" on page 11-27.
- For each import database, complete the steps in "[Adding Shared Objects to Import Databases When Creating a New Environment](#)" on page 11-28.

Configuring Populated Databases When Creating a Multiple Source Environment

After completing the steps in "[Creating a New Multiple Source Environment](#)" on page 11-25, complete the following steps for the populated databases if your environment has more than one populated database:

1. For each populated database, set the instantiation SCN at each of the other populated databases in the environment that will be a destination database of the populated source database. These instantiation SCNs must be set, and only the changes made at a particular populated database that are committed after the corresponding SCN for that database will be applied at another populated database.

For each populated database, you can set these instantiation SCNs in one of the following ways:

- a. Perform a metadata only export of the shared objects at the populated database and import the metadata at each of the other populated databases. Such an import sets the required instantiation SCNs for the populated database at the other populated databases. Make sure no rows are imported. Also, make sure the shared objects at each populated database performing a metadata import are consistent with the populated database that performed the metadata export at the time of the export.

If you are sharing DML changes only, then table level export/import is sufficient. If you are sharing DDL changes also, then additional considerations apply. See "[Setting Instantiation SCNs Using Export/Import](#)" on page 14-36 for more information about performing a metadata export/import.

- b. Set the instantiation SCNs manually at each of the other populated databases. Do this for each of the shared objects. Make sure the shared objects at each populated database are consistent with the instantiation SCNs you set at that database. See "[Setting Instantiation SCNs Using the DBMS_APPLY_ADM Package](#)" on page 14-38 for instructions.

Adding Shared Objects to Import Databases When Creating a New Environment

After completing the steps in ["Creating a New Multiple Source Environment"](#) on page 11-25, complete the following steps for the import databases:

1. Pick the populated database that you will use as the export database. Do not perform the instantiations yet.
2. For each import database, set the instantiation SCNs at all of the other databases in the environment that will be a destination database of the import database. The databases where you set the instantiation SCNs may include populated databases and other import databases.
 - a. If one or more schemas will be created at an import database during instantiation or by a subsequent shared DDL change, then run the `SET_GLOBAL_INSTANTIATION_SCN` procedure in the `DBMS_APPLY_ADM` package for this import database at all of the other databases in the environment.
 - b. If a schema exists at an import database, and one or more tables will be created in the schema during instantiation or by a subsequent shared DDL change, then run the `SET_SCHEMA_INSTANTIATION_SCN` procedure in the `DBMS_APPLY_ADM` package for the schema at all of the other databases in the environment for the import database. Do this for each such schema.

See ["Setting Instantiation SCNs at a Destination Database"](#) on page 14-35 for instructions.

Because you are running these procedures before any tables are instantiated at the import databases, and because the local capture processes are configured already for these import databases, you will not need to run the `SET_TABLE_INSTANTIATION_SCN` for each table created during the instantiation.

3. At the export database you chose in Step 1, perform an export of the shared data with the `OBJECT_CONSISTENT` export parameter set to `y`, or use a more stringent degree of consistency. Then, perform an import with the `STREAMS_INSTANTIATION` import parameter set to `y` at each import database. See ["Setting Export and Import Parameters Relevant to Streams"](#) on page 11-8 and *Oracle9i Database Utilities* for information about using Export/Import.
4. For each populated database, except for the export database, set the instantiation SCNs at each import database that will be a destination database of the populated source database. These instantiation SCNs must be set, and only the changes made at a populated database that are committed after the corresponding SCN for that database will be applied at an import database.

You can set these instantiation SCNs in one of the following ways:

- a. Perform a metadata only export at each populated database and import the metadata at each import database. Each import sets the required instantiation SCNs for the populated database at the import database. In this case, ensure that the shared objects at the import database are consistent with the populated database at the time of the export.

If you are sharing DML changes only, then table level export/import is sufficient. If you are sharing DDL changes also, then additional considerations apply. See "[Setting Instantiation SCNs Using Export/Import](#)" on page 14-36 for more information about performing a metadata export/import.

- b. For each populated database, set the instantiation SCN manually for each shared object at each import database. Make sure the shared objects at each import database are consistent with the populated database as of the corresponding instantiation SCN. See "[Setting Instantiation SCNs Using the DBMS_APPLY_ADM Package](#)" on page 14-38 for instructions.

Complete the Multiple Source Environment Configuration

Before completing the steps in this section, you should have completed the following tasks:

- "[Creating a New Multiple Source Environment](#)" on page 11-25
- "[Configuring Populated Databases When Creating a Multiple Source Environment](#)" on page 11-27, if your environment has more than one populated database
- "[Adding Shared Objects to Import Databases When Creating a New Environment](#)" on page 11-28, if your environment has one or more import databases

When all of the previous configuration steps are finished, complete the following steps:

1. Start each apply process in the environment. See "[Starting an Apply Process](#)" on page 14-7 for instructions.
2. Start each capture process the environment. See "[Starting a Capture Process](#)" on page 12-5 for instructions.

See Also: [Chapter 23, "Multiple Source Replication Example"](#) for a detailed example that creates a multiple source environment

Adding Shared Objects to an Existing Multiple Source Environment

You add existing database objects to an existing multiple source environment by adding the necessary rules to the appropriate capture processes, propagations, and apply processes.

This example uses the following terms:

- **Populated database:** A database that already contains the shared database objects being added to the multiple source environment. You must have at least one populated database to add the objects to the environment.
- **Export database:** A populated database on which you perform an export of the database objects you are adding to the environment. This export is used to instantiate the added database objects at the import databases. You may not have an export database if all of the databases in the environment are populated databases.
- **Import database:** A database that does not contain the shared database objects before they are added to the multiple source environment. You instantiate the added database objects at an import database using the export dump file from the export database. You may not have any import databases if all of the databases in the environment are populated databases.

Before creating or altering capture or propagation rules in a running Streams environment, make sure any propagations or apply processes that will receive events as a result of the new or altered rules are configured to handle these events. That is, the propagations or apply processes should exist, and each one should be associated with a rule set that handles the events appropriately. If these propagations and apply processes are not configured properly to handle these events, then events may be lost.

For example, suppose you want to add a new table to a Streams environment that already captures, propagates, and applies changes to other tables. Assume multiple capture processes in the environment will capture changes to this table, and multiple apply processes will apply changes to this table. In this case, you must add one or more table-level rules to the following rule sets:

- The rule set for each capture process that will capture changes to the table
- The rule set for each propagation that will propagate changes to the table
- The rule set for each apply process that will apply changes to the table.

If you perform administrative steps in the wrong order, you may lose events. For example, if you add the rule to the capture rule set first, without stopping the capture process, then the propagation will not propagate the changes if it does not have a rule that instructs it to do so, and the changes may be lost.

To avoid losing events, you should complete the configuration in the following order:

1. At each populated database, specify any necessary supplemental logging for the objects being added to the environment. See ["Specifying Supplemental Logging at a Source Database"](#) on page 12-9 for instructions.
2. Either stop all of the capture processes that will capture changes to the added objects, disable all of the propagation jobs that will propagate changes to the added objects, or stop all of the apply process that will apply changes to the added objects. See one of the following sections for instructions:
 - ["Stopping a Capture Process"](#) on page 12-14
 - ["Disabling a Propagation Job"](#) on page 13-17
 - ["Stopping an Apply Process"](#) on page 14-7
3. Add the relevant rules to the rule sets for the propagations and the apply processes that will propagate or apply changes to the added objects. See the following sections for instructions:
 - ["Adding Rules to the Rule Set for a Propagation"](#) on page 13-14
 - ["Adding Rules to the Rule Set for an Apply Process"](#) on page 14-8
4. Add the relevant rules to the rule set used by each capture process that will capture changes to the added objects. See ["Adding Rules to the Rule Set for a Capture Process"](#) on page 12-5 for instructions.

When you use the `DBMS_STREAMS_ADM` package to add the capture rules, it automatically runs the `PREPARE_TABLE_INSTANTIATION`, `PREPARE_SCHEMA_INSTANTIATION`, or `PREPARE_GLOBAL_INSTANTIATION` procedure in the `DBMS_CAPTURE_ADM` package for the specified table, specified schema, or entire database, respectively.

You must run the appropriate procedure to prepare for instantiation manually if any of the following conditions is true:

- You use `DBMS_RULE_ADM` to create or modify rules in a capture process rule set.
- You do not add rules for the added objects to a capture process rule set, because the capture process already captures changes to these objects. In this case, rules for the objects may be added to propagations and apply processes in the environment, but not to the capture process.

If you must prepare for instantiation manually, then see "[Preparing Database Objects for Instantiation at a Source Database](#)" on page 12-11 for instructions.

After completing these steps, complete the steps in each of the following sections that apply to your environment. You may need to complete the steps in only one of these sections or in both of these sections:

- For each populated database, complete the steps in "[Configuring Populated Databases When Adding Shared Objects](#)" on page 11-32.
- For each import database, complete the steps in "[Adding Shared Objects to Import Databases in an Existing Environment](#)" on page 11-33.

Configuring Populated Databases When Adding Shared Objects

After completing the steps in "[Adding Shared Objects to an Existing Multiple Source Environment](#)" on page 11-30, complete the following steps for each populated database:

1. For each populated database, set the instantiation SCN for each added object at the other populated databases in the environment. These instantiation SCNs must be set, and only the changes made at a particular populated database that are committed after the corresponding SCN for that database will be applied at another populated database.

For each populated database, you can set these instantiation SCNs for each added object in one of the following ways:

- a. Perform a metadata only export of the added objects at the populated database and import the metadata at each of the other populated databases. Such an import sets the required instantiation SCNs for the database at the other databases. Make sure no rows are imported. Also, make sure the shared objects at each of the other populated databases are consistent with the populated database that performed the export at the time of the export.

If you are sharing DML changes only, then table level export/import is sufficient. If you are sharing DDL changes also, then additional considerations apply. See ["Setting Instantiation SCNs Using Export/Import"](#) on page 14-36 for more information about performing a metadata export/import.

- b. Set the instantiation SCNs manually for the added objects at each of the other populated databases. Make sure every added object at each populated database is consistent with the instantiation SCNs you set at that database. See ["Setting Instantiation SCNs Using the DBMS_APPLY_ADM Package"](#) on page 14-38 for instructions.

Adding Shared Objects to Import Databases in an Existing Environment

After completing the steps in ["Adding Shared Objects to an Existing Multiple Source Environment"](#) on page 11-30, complete the following steps for the import databases:

1. Pick the populated database that you will use as the export database. Do not perform the instantiations yet.
2. For each import database, set the instantiation SCNs for the added objects at all of the other databases in the environment that will be a destination database of the import database. The databases where you set the instantiation SCNs may be populated databases and other import databases.
 - a. If one or more schemas will be created at an import database during instantiation or by a subsequent shared DDL change, then run the `SET_GLOBAL_INSTANTIATION_SCN` procedure in the `DBMS_APPLY_ADM` package for this import database at all of the other databases in the environment.
 - b. If a schema exists at an import database, and one or more tables will be created in the schema during instantiation or by a subsequent shared DDL change, then run the `SET_SCHEMA_INSTANTIATION_SCN` procedure in the `DBMS_APPLY_ADM` package for the schema for this import database at each of the other databases in the environment. Do this for each such schema.

See ["Setting Instantiation SCNs at a Destination Database"](#) on page 14-35 for instructions.

Because you are running these procedures before any tables are instantiated at the import databases, and because the local capture processes are configured already for these import databases, you will not need to run the `SET_TABLE_INSTANTIATION_SCN` for each table created during instantiation.

3. At the export database you chose in Step 1, perform an export of the added objects with the `OBJECT_CONSISTENT` export parameter set to `Y`, or use a more stringent degree of consistency. Then, perform an import of the added objects at each import database with the `STREAMS_INSTANTIATION` import parameter set to `Y`. See ["Setting Export and Import Parameters Relevant to Streams"](#) on page 11-8 and *Oracle9i Database Utilities* for information about using Export/Import.
4. For each populated database, except for the export database, set the instantiation SCNs for the added objects at each import database that will be a destination database of the populated source database. These instantiation SCNs must be set, and only the changes made at a populated database that are committed after the corresponding SCN for that database will be applied at an import database.

For each populated database, you can set these instantiation SCNs for the added objects in one of the following ways:

- a. Perform a metadata only export of the added objects at the populated database and import the metadata at each import database. Each import sets the required instantiation SCNs for the populated database at the import database. In this case, ensure that every added object at the import database is consistent with the populated database at the time of the export.

If you are sharing DML changes only, then table level export/import is sufficient. If you are sharing DDL changes also, then additional considerations apply. See ["Setting Instantiation SCNs Using Export/Import"](#) on page 14-36 for more information about performing a metadata export/import.

- b. Set the instantiation SCNs manually for the added objects at each import database. Make sure every added object at each import database is consistent with the populated database as of the corresponding instantiation SCN. See ["Setting Instantiation SCNs Using the DBMS_APPLY_ADM Package"](#) on page 14-38 for instructions.

Complete the Adding Objects to a Multiple Source Environment Configuration

Before completing the configuration, you should have completed the following tasks:

- ["Adding Shared Objects to an Existing Multiple Source Environment"](#) on page 11-30
- ["Configuring Populated Databases When Adding Shared Objects"](#) on page 11-32, if your environment had populated databases
- ["Adding Shared Objects to Import Databases in an Existing Environment"](#) on page 11-33, if your environment had import databases

When all of the previous configuration steps are finished, start each process you stopped and enable each propagation job you disabled in Step 2 on page 11-31 in ["Adding Shared Objects to an Existing Multiple Source Environment"](#). See one of the following sections for instructions:

- ["Starting a Capture Process"](#) on page 12-5
- ["Enabling a Propagation Job"](#) on page 13-11
- ["Starting an Apply Process"](#) on page 14-7

Adding a New Database to an Existing Multiple Source Environment

Complete the following steps to add a new database to an existing multiple source Streams environment:

Note: Make sure no changes are made to the objects being shared at the database you are adding to the Streams environment until the instantiation at the database is complete.

1. Complete the necessary tasks described previously in this chapter to prepare the new database for Streams:
 - ["Configuring a Streams Administrator"](#) on page 11-2
 - ["Setting Initialization Parameters Relevant to Streams"](#) on page 11-4
 - ["Configuring a Database to Run a Streams Capture Process"](#) on page 11-13
 - ["Configuring Network Connectivity and Database Links"](#) on page 11-14

Some of these tasks may not be required at the new database.

2. Create any necessary Streams queues that do not already exist. When you create a capture process or apply process, you associate the process with a specific Streams queue. When you create a propagation, you associate it with a specific source queue and destination queue. See ["Creating a Streams Queue"](#) on page 13-2 for instructions.
3. Create one or more apply processes at the new database to apply the changes from its source databases. Make sure each apply process uses a rule set that is appropriate for applying changes. Do not start any apply process at the new database. See ["Creating an Apply Process"](#) on page 14-2 for instructions.

Keeping the apply processes stopped prevents changes made at the source databases from being applied before the instantiation of the new database is completed, which would otherwise lead to incorrect data and errors.

4. If the new database will be a source database, then, at all databases that will be destination databases for the changes made at the new database, create one or more apply processes to apply changes from the new database. Make sure each apply process uses a rule set that is appropriate for applying changes. Do not start any of these new apply processes. See ["Creating an Apply Process"](#) on page 14-2 for instructions.
5. Configure propagations at the databases that will be source databases of the new database to send changes to the new database. Make sure each propagation uses a rule set that is appropriate for propagating changes. See ["Creating a Propagation"](#) on page 13-8.
6. If the new database will be a source database, then configure propagations at the new database to send changes from the new database to each of its destination databases. Make sure each propagation uses a rule set that is appropriate for propagating changes. See ["Creating a Propagation"](#) on page 13-8.
7. If the new database will be a source database, and the shared objects already exist at the new database, then specify any necessary supplemental logging for the shared objects at the new database. See ["Specifying Supplemental Logging at a Source Database"](#) on page 12-9 for instructions.
8. At each source database for the new database, prepare for instantiation each database object for which changes will be applied by an apply process at the new database. Run either the `PREPARE_TABLE_INSTANTIATION`, `PREPARE_SCHEMA_INSTANTIATION`, or `PREPARE_GLOBAL_INSTANTIATION` procedure in the `DBMS_CAPTURE_ADM` package for the specified table, specified schema, or entire database, respectively. See ["Preparing Database Objects for Instantiation at a Source Database"](#) on page 12-11 for instructions.

9. If the new database will be a source database, then create one or more capture processes to capture the relevant changes. See ["Creating a Capture Process"](#) on page 12-2 for instructions.

When you use the `DBMS_STREAMS_ADM` package to add the capture rules, it automatically runs the `PREPARE_TABLE_INSTANTIATION`, `PREPARE_SCHEMA_INSTANTIATION`, or `PREPARE_GLOBAL_INSTANTIATION` procedure in the `DBMS_CAPTURE_ADM` package for the specified table, specified schema, or entire database, respectively.

You must run the appropriate procedure to prepare for instantiation manually if any of the following conditions is true:

- You use the `DBMS_RULE_ADM` package to add or modify capture rules.
- You use an existing capture process and do not add capture rules for any shared object.

If you must prepare for instantiation manually, then see ["Preparing Database Objects for Instantiation at a Source Database"](#) on page 12-11 for instructions.

10. If the new database will be a source database, then start any capture processes you created in Step 9. See ["Starting a Capture Process"](#) on page 12-5 for instructions.

After completing these steps, complete the steps in the appropriate section:

- If the objects that are to be shared with the new database already exist at the new database, then complete the steps in ["Configuring Databases If the Shared Objects Already Exist at the New Database"](#) on page 11-38.
- If the objects that are to be shared with the new database do not already exist at the new database, complete the steps in ["Adding Shared Objects to a New Database"](#) on page 11-39.

Configuring Databases If the Shared Objects Already Exist at the New Database

After completing the steps in ["Adding a New Database to an Existing Multiple Source Environment"](#) on page 11-35, complete the following steps if the objects that are to be shared with the new database already exist at the new database:

1. For each source database of the new database, set the instantiation SCNs at the new database. These instantiation SCNs must be set, and only the changes made at a source database that are committed after the corresponding SCN for that database will be applied at the new database.

For each source database of the new database, you can set these instantiation SCNs in one of the following ways:

- a. Perform a metadata only export of the shared objects at the source database and import the metadata at the new database. The import sets the required instantiation SCNs for the source database at the new database. Make sure no rows are imported. In this case, ensure that the shared objects at the new database are consistent with the source database at the time of the export.

If you are sharing DML changes only, then table level export/import is sufficient. If you are sharing DDL changes also, then additional considerations apply. See ["Setting Instantiation SCNs Using Export/Import"](#) on page 14-36 for more information about performing a metadata export/import.

- b. Set the instantiation SCNs manually at the new database for the shared objects. Make sure the shared objects at the new database are consistent with the source database as of the corresponding instantiation SCN. See ["Setting Instantiation SCNs Using the DBMS_APPLY_ADM Package"](#) on page 14-38 for instructions.
2. For the new database, set the instantiation SCNs at each destination database of the new database. These instantiation SCNs must be set, and only the changes made at the new source database that are committed after the corresponding SCN will be applied at a destination database. If the new database is not a source database, then do not complete this step.

You can set these instantiation SCNs for the new database in one of the following ways:

- a. Perform a metadata only export at the new database and import the metadata at each destination database. Make sure no rows are imported. The import sets the required instantiation SCNs for the new database at each destination database. In this case, ensure that the shared objects at each destination database are consistent with the new database at the time of the export.

If you are sharing DML changes only, then table level export/import is sufficient. If you are sharing DDL changes also, then additional considerations apply. See "[Setting Instantiation SCNs Using Export/Import](#)" on page 14-36 for more information about performing a metadata export/import.

- b. Set the instantiation SCNs manually at each destination database for the shared objects. Make sure the shared objects at each destination database are consistent with the new database as of the corresponding instantiation SCN. See "[Setting Instantiation SCNs Using the DBMS_APPLY_ADM Package](#)" on page 14-38 for instructions.
3. Start the apply processes that you created at the new database in Step 3 on page 11-36. See "[Starting an Apply Process](#)" on page 14-7 for instructions.
4. Start the apply processes that you created at each of the other destination databases in Step 4 on page 11-36. See "[Starting an Apply Process](#)" on page 14-7 for instructions. If the new database is not a source database, then do not complete this step.

Adding Shared Objects to a New Database

After completing the steps in "[Adding a New Database to an Existing Multiple Source Environment](#)" on page 11-35, complete the following steps if the objects that are to be shared with the new database do not already exist at the new database:

1. If the new database is a source database for other databases, then, at each destination database of the new source database, set the instantiation SCNs for the new database.
 - a. If one or more schemas will be created at the new database during instantiation or by a subsequent shared DDL change, then run the `SET_GLOBAL_INSTANTIATION_SCN` procedure in the `DBMS_APPLY_ADM` package for the new database at each destination database of the new database.

- b. If a schema exists at the new database, and one or more tables will be created in the schema during instantiation or by a subsequent shared DDL change, then run the `SET_SCHEMA_INSTANTIATION_SCN` procedure in the `DBMS_APPLY_ADM` package for the schema at each destination database of the new database. Do this for each such schema.

See "[Setting Instantiation SCNs at a Destination Database](#)" on page 14-35 for instructions.

Because you are running these procedures before any tables are instantiated at the new database, and because the local capture processes are configured already at the new database, you will not need to run the `SET_TABLE_INSTANTIATION_SCN` for each table created during instantiation.

If the new database will not be a source database, then do not complete this step, and continue with the next step.

2. Pick one source database from which to instantiate the shared objects at the new database using Export/Import. First, perform the export at the source database with the `OBJECT_CONSISTENT` export parameter set to `y`, or use a more stringent degree of consistency. Then, perform the import at the new database with the `STREAMS_INSTANTIATION` import parameter set to `y`. See "[Setting Export and Import Parameters Relevant to Streams](#)" on page 11-8 and *Oracle9i Database Utilities* for information about using Export/Import.
3. For each source database of the new database, except for the source database that performed the export for instantiation in Step 2, set the instantiation SCNs at the new database. These instantiation SCNs must be set, and only the changes made at a source database that are committed after the corresponding SCN for that database will be applied at the new database.

For each source database, you can set these instantiation SCNs in one of the following ways:

- a. Perform a metadata only export at the source database and import the metadata at the new database. The import sets the required instantiation SCNs for the source database at the new database. In this case, ensure that the shared objects at the new database are consistent with the source database at the time of the export.

If you are sharing DML changes only, then table level export/import is sufficient. If you are sharing DDL changes also, then additional considerations apply. See "[Setting Instantiation SCNs Using Export/Import](#)" on page 14-36 for more information about performing a metadata export/import.

- b. Set the instantiation SCNs manually at the new database for the shared objects. Make sure the shared objects at the new database are consistent with the source database as of the corresponding instantiation SCN. See ["Setting Instantiation SCNs Using the DBMS_APPLY_ADM Package"](#) on page 14-38 for instructions.
4. Start the apply processes at the new database that you created in Step 3 on page 11-36. See ["Starting an Apply Process"](#) on page 14-7 for instructions.
5. Start the apply processes at each of the other destination databases that you created in Step 4 on page 11-36. See ["Starting an Apply Process"](#) on page 14-7 for instructions. If the new database is not a source database, then do not complete this step.

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 Streams queue.

This chapter contains these topics:

- [Creating a Capture Process](#)
- [Starting a Capture Process](#)
- [Specifying the Rule Set for a Capture Process](#)
- [Adding Rules to the Rule Set for a Capture Process](#)
- [Removing a Rule from the Rule Set for a Capture Process](#)
- [Removing the Rule Set for a Capture Process](#)
- [Setting a Capture Process Parameter](#)
- [Specifying Supplemental Logging at a Source Database](#)
- [Setting the Start SCN for a Capture Process](#)
- [Preparing Database Objects for Instantiation at a Source Database](#)
- [Aborting Preparation for Instantiation at a Source Database](#)
- [Changing the DBID of a Database Where Changes Are Captured](#)
- [Resetting the Log Sequence Number Where Changes Are Captured](#)
- [Stopping a Capture Process](#)
- [Dropping a Capture Process](#)

Each task described in this section 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 11-2

Creating a Capture Process

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

- `DBMS_STREAMS_ADM.ADD_TABLE_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 a rule set for the capture process if the capture process does not have 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 and a start SCN for the capture process.

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

- Complete the tasks described in ["Configuring a Database to Run a Streams Capture Process"](#) on page 11-13.
- Create a Streams queue to associate with the capture process, if one does not exist. See ["Creating a Streams Queue"](#) on page 13-2 for instructions.

Note: Creation of the first capture process in a database may take some time because the data dictionary is duplicated during this creation.

Example of Creating a 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 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);
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 rule set and associates it with the capture process, if the capture process does not have a rule set. 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 specifies that the capture process captures DML changes to the `hr.employees` table, and the other rule specifies that the capture process captures DDL changes to the `hr.employees` table. The rule names are specified by the system.
- Adds the two rules to the rule set associated with the capture process
- 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.

See Also:

- ["Capture Process Creation"](#) on page 2-21
- ["System-Created Rules"](#) on page 6-3
- ["Tags and Rules Created by the DBMS_STREAMS_ADM Package"](#) on page 8-3

Example of Creating a 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 capture process:

```
BEGIN
  DBMS_CAPTURE_ADM.CREATE_CAPTURE(
    queue_name    => 'strm01_queue',
    capture_name  => 'strm02_capture',
    rule_set_name => 'strmadmin.strm01_rule_set',
    start_scn     => 829381993);
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 `strm01_queue`
- Associates the capture process with an existing rule set named `strm01_rule_set`
- Specifies 829381993 as the start SCN for the capture process.

See Also:

- ["Capture Process Creation"](#) on page 2-21
- ["The Start SCN, Captured SCN, and Applied SCN for a Capture Process"](#) on page 2-15

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;
/
```

Specifying the Rule Set for a Capture Process

You specify an existing rule set that you want to associate with an existing capture process using the `rule_set_name` parameter in the `ALTER_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package. For example, the following procedure sets the 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;
/
```

See Also:

- [Chapter 5, "Rules"](#)
- [Chapter 6, "How Rules Are Used In Streams"](#)

Adding Rules to the Rule Set for a Capture Process

To add rules to the 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_SCHEMA_RULES`
- `DBMS_STREAMS_ADM.ADD_GLOBAL_RULES`

The following is an example that runs the `ADD_TABLE_RULES` procedure in the `DBMS_STREAMS_ADM` package to add rules to the 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);
END;
/
```

Running this procedure performs the following actions:

- Creates two rules. One rule specifies that the capture process captures DML changes to the `hr.departments` table, and the other rule specifies that the capture process captures DDL changes to the `hr.departments` table. The rule names are specified by the system.
- Adds the two rules to the rule set associated with the capture process

See Also: ["System-Created Rules"](#) on page 6-3

Removing a Rule from the Rule Set for a Capture Process

You specify that you want to remove a rule from the 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 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);
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.

In addition, if you want to remove all of the rules in the rule set for the capture process, then specify `NULL` for the `rule_name` parameter when you run the `REMOVE_RULE` procedure.

Note: If you drop all of the rules in the rule set for a capture process, then the capture process captures no events.

Removing the Rule Set for a Capture Process

You specify that you want to remove the rule set from an existing capture process by setting the `remove_rule_set` parameter to `true` in the `ALTER_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package. For example, the following procedure removes the rule set from a capture process named `strm01_capture`.

```
BEGIN
  DBMS_CAPTURE_ADM.ALTER_CAPTURE(
    capture_name => 'strm01_capture',
    remove_rule_set => true);
END;
/
```

Note: If you remove a rule set for a capture process, then the capture process captures all supported changes to all objects in the database, excluding database objects in the `SYS` and `SYSTEM` schemas.

Setting a Capture Process Parameter

You 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 2-25
- The `DBMS_CAPTURE_ADM.SET_PARAMETER` procedure in the *Oracle9i Supplied PL/SQL Packages and Types Reference* for detailed information about the capture process parameters

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. This section illustrates how to specify supplemental logging at a source database.

Note: LOBs, LONGs, and user-defined types cannot be part of a supplemental log group

See Also: ["Supplemental Logging in a Streams Environment"](#) on page 2-11 for information about when supplemental logging is required

Specifying Table Supplemental Logging Using Unconditional Log Groups

To specify an unconditional supplemental log group, you must create redo log groups that include the necessary columns using the `ADD SUPPLEMENTAL LOG GROUP` clause and the `ALWAYS` specification in an `ALTER TABLE` statement. These redo log groups can include key columns, if necessary.

For example, the following statement adds the primary key column of the `hr.departments` table to an unconditional log group named `log_group_dep_pk`:

```
ALTER TABLE hr.departments ADD SUPPLEMENTAL LOG GROUP log_group_dep_pk
(department_id) ALWAYS;
```

The `ALWAYS` specification makes this log group an unconditional log group.

Specifying Table Supplemental Logging Using Conditional Log Groups

To specify a conditional supplemental log group, you must create redo log groups that include the necessary columns using the `ADD SUPPLEMENTAL LOG GROUP` clause in the `ALTER TABLE` statement. To make the log group conditional, do not include the `ALWAYS` specification.

For example, suppose the `min_salary` and `max_salary` columns in the `hr.jobs` table are included in a column list for conflict resolution at a destination database. The following statement adds the `min_salary` and `max_salary` columns to a log conditional group named `log_group_jobs_cr`:

```
ALTER TABLE hr.jobs ADD SUPPLEMENTAL LOG GROUP log_group_jobs_cr
(min_salary, max_salary);
```

Dropping a Supplemental Log Group

To drop a conditional or unconditional supplemental log group, use the `DROP SUPPLEMENTAL LOG GROUP` clause in the `ALTER TABLE` statement. For example, to drop a supplemental log group named `log_group_jobs_cr`, run the following statement:

```
ALTER TABLE hr.jobs DROP SUPPLEMENTAL LOG GROUP log_group_jobs_cr;
```

Specifying Database Supplemental Logging of Key Columns

You also have the option of specifying supplemental logging for all primary key and unique key columns in a source database. You may choose this option if you configure a capture process to capture changes to an entire database. To specify supplemental logging for all primary key and unique key columns in a source database, issue the following SQL statement:

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY, UNIQUE INDEX) COLUMNS;
```

If your primary and unique key columns are the same at all source and destination databases, then running this command at the source database provides the supplemental logging needed for primary and unique key columns at all destination databases.

Dropping Database Supplemental Logging of Key Columns

To drop supplemental logging for all primary key and unique key columns in a source database, issue the following SQL statement:

```
ALTER DATABASE DROP SUPPLEMENTAL LOG DATA;
```

Note: Dropping database supplemental logging of key columns does not affect any existing table-level supplemental log groups.

Setting the Start SCN for a Capture Process

You specify the start SCN for an existing capture process using the `start_scn` parameter in the `ALTER_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package. The SCN value specified must be from a point in time after the first capture process was created for the database. The first capture process for the database may or may not be the capture process being altered. An error is returned if an invalid SCN is specified. Typically, you reset a start SCN for a capture process if point-in-time recovery must be performed on one of the destination databases for changes from the capture process.

For example, the following procedure sets the start SCN for a capture process named `strm01_capture` to 750338948.

```
BEGIN
  DBMS_CAPTURE_ADM.ALTER_CAPTURE(
    capture_name => 'strm01_capture',
    start_scn    => 750338948);
END;
/
```

See Also:

- ["The Start SCN, Captured SCN, and Applied SCN for a Capture Process"](#) on page 2-15
- ["Performing Database Point-in-Time Recovery on a Destination Database"](#) on page 16-29

Preparing Database Objects for Instantiation at a Source Database

The following procedures in the `DBMS_CAPTURE_ADM` package prepare database objects for instantiation:

- `PREPARE_TABLE_INSTANTIATION` prepares a single table for instantiation.
- `PREPARE_SCHEMA_INSTANTIATION` prepares for instantiation all of the database objects in a schema and all database 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.

If you run one of these procedures while a long running transaction is modifying one or more database objects being prepared for instantiation, then the procedure

will wait until the long running transaction is complete before it records the lowest SCN.

For example, to prepare the `hr.regions` table for instantiation, run the following procedure:

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

See Also:

- ["Instantiation"](#) on page 2-13
- ["Streams Data Dictionary for Propagations"](#) on page 3-25
- ["Streams Data Dictionary for an Apply Process"](#) on page 4-33
- ["Configuring a Capture-Based Streams Environment"](#) on page 11-15

Aborting Preparation for Instantiation at a Source Database

The following procedures in the `DBMS_CAPTURE_ADM` package abort preparation for instantiation:

- `ABORT_TABLE_INSTANTIATION` reverses the effects of `PREPARE_TABLE_INSTANTIATION`.
- `ABORT_SCHEMA_INSTANTIATION` reverses the effects of `PREPARE_SCHEMA_INSTANTIATION`.
- `ABORT_GLOBAL_INSTANTIATION` reverses the effects of `PREPARE_GLOBAL_INSTANTIATION`.

These procedures remove data dictionary information related to the potential instantiation of the relevant database objects.

For example, to abort the preparation for instantiation of the `hr.regions` table, run the following procedure:

```
BEGIN
  DBMS_CAPTURE_ADM.ABORT_TABLE_INSTANTIATION(
    table_name => 'hr.regions');
END;
/
```

Changing the DBID of a Database Where Changes Are Captured

Typically, database administrators change the DBID of a database when it is a clone of another database. You can view the DBID for a database by querying the DBID column in the V\$DATABASE dynamic performance view.

If a capture process is capturing changes generated by a database for which you have changed the DBID, then complete the following steps:

1. Shut down the database.
2. Restart the database with RESTRICTED SESSION enabled using STARTUP RESTRICT.
3. Drop the capture process.
4. Run the ALTER SYSTEM SWITCH LOGFILE statement on the database.
5. If the database has captured any changes, then manually resynchronize the data at all destination databases that apply changes originating at this source database. If the database never captured any changes, then this step is not necessary.
6. Recreate the capture process, if necessary.
7. Disable the restricted session using the ALTER SYSTEM DISABLE RESTRICTED SESSION statement.

See Also: *Oracle9i Database Utilities* for more information about changing the DBID of a database using the DBNEWID utility

Resetting the Log Sequence Number Where Changes Are Captured

Typically, database administrators reset the log sequence number of a database during point-in-time recovery. The ALTER DATABASE OPEN RESETLOGS statement is an example of a statement that resets the log sequence number. When you reset the log sequence number of a database, any existing local capture processes become unusable.

If a capture process is capturing changes generated by a database for which you have reset the log sequence number, then complete the following steps:

1. Drop the capture process.
2. Manually resynchronize the data at all destination databases that apply changes originating at this source database.
3. Recreate the capture process, if necessary.

See Also: *Oracle9i Backup and Recovery Concepts* for more information about point-in-time recovery

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 `strm01_capture`:

```
BEGIN
  DBMS_CAPTURE_ADM.DROP_CAPTURE(
    capture_name => 'strm01_capture');
END;
/
```

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

Managing Staging and Propagation

This chapter provides instructions for managing Streams queues, propagations, and messaging environments.

This chapter contains these topics:

- [Managing Streams Queues](#)
- [Managing Streams Propagations and Propagation Jobs](#)
- [Managing a Streams Messaging Environment](#)

Each task described in this section 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 11-2

Managing Streams Queues

A Streams queue stages events whose payloads are of `SYS.AnyData` type. Therefore, a Streams 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 Streams queue, and each Streams propagation is associated with one Streams source queue and one Streams destination queue.

This section provides instructions for completing the following tasks related to Streams queues:

- [Creating a Streams Queue](#)
- [Enabling a User to Perform Operations on a Secure Queue](#)
- [Disabling a User from Performing Operations on a Secure Queue](#)
- [Dropping a Streams Queue](#)

Creating a Streams Queue

You use the `SET_UP_QUEUE` procedure in the `DBMS_STREAMS_ADM` package to create a Streams queue. This procedure enables you to specify the following for the Streams 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 Streams queue named `strm01_queue` 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 => 'strm01_queue_table',
    queue_name  => 'strm01_queue',
    queue_user  => 'hr' );
END;
/
```

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

See Also:

- ["Wrapping User Message Payloads in a SYS.AnyData Wrapper"](#) on page 13-19 for an example that creates a `SYS.AnyData` queue using procedures in the `DBMS_AQADM` package
- ["Secure Queues"](#) on page 3-22
- ["Transactional and Nontransactional Queues"](#) on page 3-24

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 users with an agent manually

The following example illustrates associating a user with an agent manually. Suppose you want to enable the `oe` user to perform queue operations on the `strm01_queue` created in ["Creating a Streams Queue"](#) on page 13-2. The following steps configure the `oe` user as a secure queue user of `strm01_queue`:

1. Connect as an administrative user who can create 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);
  SYS.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_AQ` package, if the user is not already granted this privilege.

```
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 3-22
- *Oracle9i Supplied 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 13-3, 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 Streams Queue"](#) on page 13-2.

Attention: If an agent is used for multiple secure queues, then running `DISABLE_DB_ACCESS` for the agent prevents the user 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 3-22
- *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about AQ agents and using the `DBMS_AQADM` package

Dropping a Streams Queue

To drop an existing Streams queue, perform the same actions that you would to drop a typed queue. A Streams queue may be dropped in the following ways:

- Explicitly, either by using the `STOP_QUEUE`, `DROP_QUEUE`, and `DROP_QUEUE_TABLE` procedures in the `DBMS_AQADM` package, or by using only the `DROP_QUEUE_TABLE` procedure on the queue table containing the Streams queue with the `force` parameter set to `true`
- Dropping the user who owns the queue with the `CASCADE` option

When you drop a Streams queue, all of the error transactions that were moved to the exception queue from the Streams queue are deleted automatically.

See Also: *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about dropping queues

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 the 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: *Oracle9i Application Developer's Guide - Advanced Queuing* 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_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 a rule set for the propagation if the propagation does not have 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. 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 Streams Queue](#)" on page 13-2 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 11-14 for information.

Example of Creating a Propagation 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 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' );
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 rule set and associates it with the propagation, if the propagation does not have a rule set. 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 specifies that the propagation propagates row LCRs that contain the results of DML changes to the `hr.departments` table, and the other rule specifies that the propagation propagates DDL LCRs that contain changes to the `hr.departments` table. The rule names are specified by the system.
- Adds the two rules to the rule set associated with the propagation
- 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 of the LCRs to be propagated is `db1.net`, which may or may not be the current database
- Creates a propagation job, if one does not exist for the specified database link

See Also:

- ["Event Propagation Between Queues"](#) on page 3-4
- ["System-Created Rules"](#) on page 6-3
- ["Tags and Rules Created by the DBMS_STREAMS_ADM Package"](#) on page 8-3

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 => 'db2.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 `db2.net` database. Depending on the rules in the rule set, the propagated events may be captured events or user-enqueued events, or both.
- Specifies that the propagation uses the `db2.net` database link to propagate the events
- Associates the propagation with an existing rule set named `strm01_rule_set`
- 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 3-4

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:

- *Oracle9i Application Developer's Guide - Advanced Queuing* for more information about using the `ENABLE_PROPAGATION_SCHEDULE` procedure
- ["Propagation Jobs"](#) on page 3-20

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:

- ["Unschedulering a Propagation Job"](#) on page 13-13
- *Oracle9i Application Developer's Guide - Advanced Queuing* for more information about using the `SCHEDULE_PROPAGATION` procedure
- ["Propagation Jobs"](#) on page 3-20

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:

- *Oracle9i Application Developer's Guide - Advanced Queuing* for more information about using the `ALTER_PROPAGATION_SCHEDULE` procedure
- "[Propagation Jobs](#)" on page 3-20

Unscheduled 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 13-11
- *Oracle9i Application Developer's Guide - Advanced Queuing* for more information about using the `SCHEDULE_PROPAGATION` procedure
- ["Propagation Jobs"](#) on page 3-20

Specifying the Rule Set for a Propagation

You specify the rule set that you want to associate with a propagation using the `rule_set_name` parameter in the `ALTER_PROPAGATION` procedure in the `DBMS_PROPAGATION_ADM` package. For example, the following procedure sets the 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;
/
```

See Also:

- [Chapter 5, "Rules"](#)
- [Chapter 6, "How Rules Are Used In Streams"](#)

Adding Rules to the Rule Set for a Propagation

You 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_SCHEMA_PROPAGATION_RULES`
- `DBMS_STREAMS_ADM.ADD_GLOBAL_PROPAGATION_RULES`

The following is an example that runs the `ADD_TABLE_RULES` procedure in the `DBMS_STREAMS_ADM` package to add rules to the rule set of a 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' );
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 two rules. One rule specifies that the propagation propagates row LCRs that contain the results of DML changes to the `hr.locations` table, and the other rule specifies that the propagation propagates DDL LCRs that contain changes to the `hr.locations` table. The rule names are specified by the system.
- Adds the two rules to the rule set associated with the propagation
- Specifies that the source database of the LCRs to be propagated is `dbs1.net`, which may or may not be the current database

See Also:

- ["Event Propagation Between Queues"](#) on page 3-4
- ["System-Created Rules"](#) on page 6-3

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 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);
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.

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.

Note: If you drop all of the rules in the rule set for a propagation, then the propagation propagations no events in the source queue to the destination queue.

Removing the Rule Set for a Propagation

You specify that you want to remove the rule set from a propagation by setting the `rule_set_name` parameter to `NULL` in the `ALTER_PROPAGATION` procedure in the `DBMS_PROPAGATION_ADM` package. For example, the following procedure removes the rule set from a propagation named `strm01_propagation`.

```
BEGIN
  DBMS_PROPAGATION_ADM.ALTER_PROPAGATION(
    propagation_name => 'strm01_propagation',
    rule_set_name    => NULL);
END;
/
```

Note: If you remove a rule set for a propagation, 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: *Oracle9i Application Developer's Guide - Advanced Queuing* 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');
END;
/
```

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](#)
- [Propagating Messages Between a `SYS.AnyData` Queue and a Typed Queue](#)

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 3-11 for conceptual information about messaging in Streams
- ["Configuring a Streams Administrator"](#) on page 11-2
- [Chapter 19, "Streams Messaging Example"](#)
- *Oracle9i Application Developer's Guide - Advanced Queuing* for more information about AQ
- *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about the `SYS.AnyData` type

Wrapping User Message Payloads in a SYS.AnyData Wrapper

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

Example of Wrapping a Payload in a SYS.AnyData Payload and Enqueueing It

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 `dbssl.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@dbssl.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.

5. Create an agent:

```
EXEC DBMS_AQADM.CREATE_AQ_AGENT(agent_name => 'local_agent');
```

6. Add a subscriber to the `oe_q_any` queue. This subscriber will perform explicit dequeues of events.

```
DECLARE
  subscriber SYS.AQ$_AGENT;
BEGIN
  subscriber := SYS.AQ$_AGENT('LOCAL_AGENT', NULL, NULL);
  SYS.DBMS_AQADM.ADD_SUBSCRIBER(
    queue_name => 'strmadmin.oe_q_any',
    subscriber => subscriber);
END;
/
```

7. Associate the `oe` user with the `local_agent` agent:

```
BEGIN
  DBMS_AQADM.ENABLE_DB_ACCESS(
    agent_name => 'local_agent',
    db_username => 'oe');
END;
/
```

8. Connect as the `oe` user.

```
CONNECT oe/oe@dbs1.net
```

9. 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('LOCAL_AGENT', NULL, NULL);
    DBMS_AQ.ENQUEUE(
        queue_name          => 'strmadmin.oe_q_any',
        enqueue_options     => enqopt,
        message_properties  => mprop,
        payload              => payload,
        msgid                => enq_msgid);
END;
/
```

10. Run the procedure you created in Step 9 by specifying the appropriate `Convertdata_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 17-13 for information about viewing the contents of these enqueued messages

Example of 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 ["Example of Wrapping a Payload in a SYS.AnyData Payload and Enqueuing It"](#) on page 13-19.

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@dbssl.net

SELECT MSG_ID, MSG_STATE, CONSUMER_NAME FROM AQ$OE_Q_TABLE_ANY;
```

1. Connect as the oe user:

```
CONNECT oe/oe@dbssl.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('LOCAL_AGENT');

```

Propagating Messages Between a SYS.AnyData Queue and a Typed Queue

`SYS.AnyData` queues can interoperate with typed queues in a Streams environment. A typed queue is a queue that can stage messages of a particular type only. To propagate a message from a `SYS.AnyData` queue to a typed queue, the message must be transformed to match the type of the typed queue. The following sections provide examples of propagating non-LCR user messages and LCRs between a `SYS.AnyData` queue and a typed queue.

Note: The examples in this section assume that you have completed the examples in "[Wrapping User Message Payloads in a SYS.AnyData Wrapper](#)" on page 13-19.

See Also: "[Message Propagation and SYS.AnyData Queues](#)" on page 3-16 for more information about propagation between `SYS.AnyData` and typed queues

Example of Propagating Non-LCR User Messages to a Typed Queue

The following steps set up propagation from a `SYS.AnyData` queue named `oe_q_any` to a typed queue of type `oe.cust_address_typ` named `oe_q_address`. The source queue `oe_q_any` is at the `dbs1.net` database, and the destination queue `oe_q_address` is at the `dbs2.net` database. Both queues are owned by `strmadmin`.

1. Connect as an administrative user who can grant privileges at `dbs1.net`.
2. Grant the following privilege to `strmadmin`, if it was not already granted.

```
GRANT EXECUTE ON DBMS_TRANSFORM TO strmadmin;
```

3. Grant `strmadmin` EXECUTE privilege on `oe.cust_address_typ` at `dbs1.net` and `dbs2.net`.

```
CONNECT oe/oe@dbs1.net
```

```
GRANT EXECUTE ON oe.cust_address_typ TO strmadmin;
```

```
CONNECT oe/oe@dbs2.net
```

```
GRANT EXECUTE ON oe.cust_address_typ TO strmadmin;
```


4. Create a typed queue at `dbs2.net`, if one does not already exist.

```
CONNECT strmadmin/strmadminpw@dbs2.net

BEGIN
  DBMS_AQADM.CREATE_QUEUE_TABLE(
    queue_table      => 'strmadmin.oe_q_table_address',
    queue_payload_type => 'oe.cust_address_typ',
    multiple_consumers => true);
  DBMS_AQADM.CREATE_QUEUE(
    queue_name      => 'strmadmin.oe_q_address',
    queue_table     => 'strmadmin.oe_q_table_address');
  DBMS_AQADM.START_QUEUE(
    queue_name      => 'strmadmin.oe_q_address');
END;
/
```

5. Create a database link between `dbs1.net` and `dbs2.net` if one does not already exist.

```
CONNECT strmadmin/strmadminpw@dbs1.net

CREATE DATABASE LINK dbs2.net CONNECT TO strmadmin IDENTIFIED BY strmadminpw
  USING 'DBS2.NET';
```

6. Create a function called `any_to_cust_address_typ` in the `strmadmin` schema at `dbs1.net` that takes a `SYS.AnyData` payload containing a `oe.cust_address_typ` object and returns the `oe.cust_address_typ` object.

```
CREATE OR REPLACE FUNCTION strmadmin.any_to_cust_address_typ(
  in_any IN SYS.AnyData)
RETURN OE.CUST_ADDRESS_TYP
AS
  address      OE.CUST_ADDRESS_TYP;
  num_var      NUMBER;
  type_name    VARCHAR2(100);
BEGIN
  -- Get the type of object
  type_name := in_any.GetTypeName();
  -- Check if the object type is OE.CUST_ADDRESS_TYP
  IF (type_name = 'OE.CUST_ADDRESS_TYP') THEN
    -- Put the address in the message into the address variable
    num_var := in_any.GetObject(address);
    RETURN address;
  END IF;
END;
```

```

ELSE
    raise_application_error(-20101, 'Conversion failed - ' || type_name);
END IF;
END;
/

```

7. Create a transformation at dbs1.net using the DBMS_TRANSFORM package.

```

BEGIN
    DBMS_TRANSFORM.CREATE_TRANSFORMATION(
        schema      => 'strmadmin',
        name        => 'anytoaddress',
        from_schema => 'SYS',
        from_type   => 'ANYDATA',
        to_schema   => 'oe',
        to_type     => 'cust_address_typ',
        transformation => 'strmadmin.any_to_cust_address_typ(source.user_data)');
END;
/

```

8. Create a subscriber for the typed queue if one does not already exist. The subscriber must contain a rule that ensures that only messages of the appropriate type are propagated to the destination queue.

```

DECLARE
    subscriber SYS.AQ$_AGENT;
BEGIN
    subscriber := SYS.AQ$_AGENT ('ADDRESS_AGENT_REMOTE',
                                'STRMADMIN.OE_Q_ADDRESS@DBS2.NET',
                                0);

    DBMS_AQADM.ADD_SUBSCRIBER(
        queue_name      => 'strmadmin.oe_q_any',
        subscriber      => subscriber,
        rule            =>
            'TAB.USER_DATA.GetTypeName()='OE.CUST_ADDRESS_TYP'',
        transformation => 'strmadmin.anytoaddress');
END;
/

```

9. Schedule propagation between the `SYS.AnyData` queue at `dbs1.net` and the typed queue at `dbs2.net`.

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

10. Enqueue a message of `oe.cust_address_typ` type wrapped in a `SYS.AnyData` wrapper:

```
CONNECT oe/oe@dbs1.net

BEGIN
  oe.eng_proc(SYS.AnyData.ConvertObject(oe.cust_address_typ(
    '1668 Chong Tao', '111181', 'Beijing', NULL, 'CN')));
END;
/
COMMIT;
```

11. After allowing some time for propagation, query the queue table at `dbs2.net` to view the propagated message:

```
CONNECT strmadmin/strmadminpw@dbs2.net

SELECT MSG_ID, MSG_STATE, CONSUMER_NAME FROM AQ$OE_Q_TABLE_ADDRESS;
```

See Also: *Oracle9i Application Developer's Guide - Advanced Queuing* for more information about transformations during propagation

Example of Propagating LCRs to a Typed Queue

To propagate LCRs from a `SYS.AnyData` queue to a typed queue, you complete the same steps as you do for non-LCR events, but Oracle supplies the transformation functions. You can use the following functions in the `DBMS_STREAMS` package to transform LCRs in `SYS.AnyData` queues to messages in typed queues:

- The `CONVERT_ANYDATA_TO_LCR_ROW` function transforms `SYS.AnyData` payload containing a row LCR into `SYS.LCR$_ROW_RECORD` payload.
- The `CONVERT_ANYDATA_TO_LCR_DDL` function transforms `SYS.AnyData` payload containing a DDL LCR into `SYS.LCR$_DDL_RECORD` payload.

You can propagate user-enqueued LCRs to an appropriate typed queue, but propagation of captured LCRs to a typed queue is not supported.

The following example sets up propagation of row LCRs from a `SYS.AnyData` queue named `oe_q_any` to a typed queue of type `SYS.LCR$_ROW_RECORD` named `oe_q_lcr`. The source queue `oe_q_any` is at the `dbs1.net` database, and the destination queue `oe_q_lcr` is at the `dbs3.net` database.

1. Connect as an administrative user who can grant privileges at `dbs1.net`.
2. Grant the following privilege to `strmadmin`, if it was not already granted.

```
GRANT EXECUTE ON DBMS_TRANSFORM TO strmadmin;
```

3. Create a queue of the LCR type if one does not already exist.

```
CONNECT strmadmin/strmadminpw@dbs3.net

BEGIN
  DBMS_AQADM.CREATE_QUEUE_TABLE(
    queue_table      => 'strmadmin.oe_q_table_lcr',
    queue_payload_type => 'SYS.LCR$_ROW_RECORD',
    multiple_consumers => true);
  DBMS_AQADM.CREATE_QUEUE(
    queue_name      => 'strmadmin.oe_q_lcr',
    queue_table     => 'strmadmin.oe_q_table_lcr');
  DBMS_AQADM.START_QUEUE(
    queue_name      => 'strmadmin.oe_q_lcr');
END;
/
```

4. Create a database link between `dbs1.net` and `dbs3.net` if one does not already exist.

```
CONNECT strmadmin/strmadminpw@dbs1.net

CREATE DATABASE LINK dbs3.net CONNECT TO strmadmin IDENTIFIED BY strmadminpw
  USING 'DBS3.NET';
```

5. Create a transformation at `dbs1.net` using the `DBMS_TRANSFORM` package.

```
BEGIN
  DBMS_TRANSFORM.CREATE_TRANSFORMATION(
    schema      => 'strmadmin',
    name        => 'anytolcr',
    from_schema => 'SYS',
    from_type   => 'ANYDATA',
    to_schema   => 'SYS',
    to_type     => 'LCR$_ROW_RECORD',
    transformation =>
      'SYS.DEMS_STREAMS.CONVERT_ANYDATA_TO_LCR_ROW(source.user_data)');
END;
/
```

6. Create a subscriber at the typed queue if one does not already exist. The subscriber specifies the `CONVERT_ANYDATA_TO_LCR_ROW` function for the transformation parameter.

```
DECLARE
  subscriber SYS.AQ$_AGENT;
BEGIN
  subscriber := SYS.AQ$_AGENT (
    'ROW_LCR_AGENT_REMOTE',
    'STRMADMIN.OE_Q_LCR@DBS3.NET',
    0);
  DBMS_AQADM.ADD_SUBSCRIBER(
    queue_name  => 'strmadmin.oe_q_any',
    subscriber  => subscriber,
    rule        => 'TAB.USER_DATA.GetTypeName()='SYS.LCR$_ROW_RECORD'',
    transformation => 'strmadmin.anytolcr');
END;
/
```

7. Schedule propagation between the SYS.AnyData queue at dbs1.net and the LCR queue at dbs3.net.

```
BEGIN
  DBMS_AQADM.SCHEDULE_PROPAGATION(
    queue_name => 'strmadmin.oe_q_any',
    destination => 'dbs3.net');
END;
/
```

8. Create a procedure to construct and enqueue a row LCR into the strmadmin.oe_q_any queue:

```
CONNECT oe/oe@dbs1.net

CREATE OR REPLACE PROCEDURE oe.enq_row_lcr_proc(
  source_dbname  VARCHAR2,
  cmd_type       VARCHAR2,
  obj_owner      VARCHAR2,
  obj_name       VARCHAR2,
  old_vals       SYS.LCR$_ROW_LIST,
  new_vals       SYS.LCR$_ROW_LIST) AS
  eopt           DBMS_AQ.ENQUEUE_OPTIONS_T;
  mprop          DBMS_AQ.MESSAGE_PROPERTIES_T;
  enq_msgid      RAW(16);
  row_lcr        SYS.LCR$_ROW_RECORD;
BEGIN
  mprop.SENDER_ID := SYS.AQ$_AGENT('LOCAL_AGENT', NULL, NULL);
  -- Construct the LCR based on information passed to procedure
  row_lcr := SYS.LCR$_ROW_RECORD.CONSTRUCT(
    source_database_name => source_dbname,
    command_type         => cmd_type,
    object_owner         => obj_owner,
    object_name          => obj_name,
    old_values           => old_vals,
    new_values           => new_vals);
  -- Enqueue the created row LCR
  DBMS_AQ.ENQUEUE(
    queue_name           => 'strmadmin.oe_q_any',
    enqueue_options     => eopt,
    message_properties  => mprop,
    payload              => SYS.AnyData.ConvertObject(row_lcr),
    msgid               => enq_msgid);
END enq_row_lcr_proc;
/
```

9. Create a row LCR that inserts a row into the `oe.inventories` table and enqueue the row LCR into the `strmadmin.oe_q_any` queue.

```

DECLARE
  newunit1 SYS.LCR$_ROW_UNIT;
  newunit2 SYS.LCR$_ROW_UNIT;
  newunit3 SYS.LCR$_ROW_UNIT;
  newvals  SYS.LCR$_ROW_LIST;
BEGIN
  newunit1 := SYS.LCR$_ROW_UNIT(
    'PRODUCT_ID',
    SYS.AnyData.ConvertNumber(3503),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  newunit2 := SYS.LCR$_ROW_UNIT(
    'WAREHOUSE_ID',
    SYS.AnyData.ConvertNumber(1),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  newunit3 := SYS.LCR$_ROW_UNIT(
    'QUANTITY_ON_HAND',
    SYS.AnyData.ConvertNumber(157),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  newvals := SYS.LCR$_ROW_LIST(newunit1,newunit2,newunit3);
  oe.enq_row_lcr_proc(
    source_dbname => 'DBS1.NET',
    cmd_type      => 'INSERT',
    obj_owner     => 'OE',
    obj_name      => 'INVENTORIES',
    old_vals      => NULL,
    new_vals      => newvals);
END;
/
COMMIT;

```

10. After allowing some time for propagation, query the queue table at `db3.net` to view the propagated message:

```
CONNECT strmadmin/strmadminpw@db3.net
```

```
SELECT MSG_ID, MSG_STATE, CONSUMER_NAME FROM AQ$OE_Q_TABLE_LCR;
```

See Also: The `DBMS_STREAMS` package in the *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about the row LCR and DDL LCR conversion functions

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:

- [Creating, Starting, Stopping, and Dropping an Apply Process](#)
- [Managing the Rule Set for an Apply Process](#)
- [Setting an Apply Process Parameter](#)
- [Setting the Apply User for an Apply Process](#)
- [Managing the Message Handler for an Apply Process](#)
- [Managing a DML Handler](#)
- [Managing the DDL Handler for an Apply Process](#)
- [Managing an Error Handler](#)
- [Managing the Substitute Key Columns for a Table](#)
- [Managing Streams Conflict Resolution](#)
- [Managing Apply Errors](#)
- [Setting Instantiation SCNs at a Destination Database](#)

Each task described in this section 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 11-2](#)
- ["Managing Streams Tags for an Apply Process" on page 16-27](#)

Creating, Starting, Stopping, and Dropping an Apply Process

This section contains instructions for creating, starting, stopping and 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_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 a rule set for the apply process if the apply process does not have a rule set, and may add table, schema, or global rules to the rule set.

The `CREATE_APPLY` procedure 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 and a number of other options, such as event handlers, an apply user, an apply tag, and whether to apply captured or user-enqueued events.

Before you create an apply process, create a Streams 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.

See Also:

- ["Creating a Streams Queue" on page 13-2](#)
- ["Supplemental Logging in a Streams Environment" on page 2-11 for information about when supplemental logging is required](#)
- ["Specifying Supplemental Logging at a Source Database" on page 12-9](#)

Example of Creating an Apply Process 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:

```
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');
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 rule set and associates it with the apply process, if the apply process does not have a rule set. The rule set uses the `SYS.STREAMS$_EVALUATION_CONTEXT` evaluation context. The rule set name is specified by the system.
- Creates one rule that specifies that the apply process applies 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 rule set associated with the apply process
- 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.

See Also:

- ["Apply Process Creation"](#) on page 4-32
- ["System-Created Rules"](#) on page 6-3
- ["Tags and Rules Created by the DBMS_STREAMS_ADM Package"](#) on page 8-3

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.

See Also:

- ["Apply Process Creation"](#) on page 4-32
- ["Event Processing Options"](#) on page 4-4 for more information about event handlers
- ["Tags and an Apply Process"](#) on page 8-6
- ["Oracle to Non-Oracle Data Sharing with Streams"](#) on page 9-2 for information about configuring an apply process to apply events to a non-Oracle database using the `apply_database_link` parameter

Example of Creating an Apply Process to Apply Captured Events 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          => 'strm02_queue',
    apply_name          => 'strm02_apply',
    rule_set_name       => 'strmadmin.strm01_rule_set',
    message_handler     => NULL,
    ddl_handler         => 'hr.ddl_handler',
    apply_user          => 'hr',
    apply_database_link => NULL,
    apply_tag           => HEXTORAW('5'),
    apply_captured      => true);
END;
/
```

Running this procedure performs the following actions:

- Creates an apply process named `strm02_apply`. An apply process with the same name must not exist.
- Associates the apply process with an existing queue named `strm02_queue`
- Associates the apply process with an existing rule set named `strm01_rule_set`
- Specifies that the apply process does not use a message handler.
- Specifies that the DDL handler is the `ddl_handler` PL/SQL procedure in the `hr` schema. The user who runs the `CREATE_APPLY` procedure must have `EXECUTE` privilege on the `ddl_handler` PL/SQL 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 the capture process, is staged in the queue for the apply process, then this apply process does not apply the LCR.

Example of Creating an Apply Process to Apply User-Enqueued Events 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          => 'strm01_queue',
    apply_name          => 'strm03_apply',
    rule_set_name       => 'strmadmin.strm02_rule_set',
    message_handler     => 'strmadmin.mes_handler',
    ddl_handler         => NULL,
    apply_user          => NULL,
    apply_database_link => NULL,
    apply_tag           => NULL,
    apply_captured      => false);
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 `strm01_queue`
- Associates the apply process with an existing rule set named `strm02_rule_set`
- 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.

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');
END;
/
```

An error is raised if you try to drop an apply process and there are errors in the exception queue for the specified apply process. Therefore, if there are errors in the exception queue for an apply process, delete the errors before dropping the apply process.

See Also: ["Managing Apply Errors"](#) on page 14-33

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 the 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 specify the rule set that you want to associate with an apply process using the `rule_set_name` parameter in the `ALTER_APPLY` procedure in the `DBMS_APPLY_ADM` package. For example, the following procedure sets the 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;
/
```

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`

The following is an example that runs the `ADD_TABLE_RULES` procedure in the `DBMS_STREAMS_ADM` package to add rules to the 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');
END;
/
```

Running this procedure performs the following actions:

- Creates one rule that specifies that the apply process applies 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 specifies that the apply process applies DDL LCRs that contain the results of DDL changes to the `hr.departments` table. The rule name is specified by the system.
- Adds the rule to the rule set associated with the apply process
- Specifies that the apply process applies LCRs only from the `dbs1.net` source database.

See Also: ["System-Created Rules"](#) on page 6-3

Removing a Rule from the Rule Set for an Apply Process

You specify that you want to remove a rule from the 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 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);
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.

In addition, if you want to remove all of the rules in the rule set for the apply process, then specify `NULL` for the `rule_name` parameter when you run the `REMOVE_RULE` procedure.

Note: If you drop all of the rules in the rule set for an apply process that applies captured events, then the apply process does not apply any captured events in its queue. Similarly, if you drop all of the rules in the rule set for an apply process that applies user-enqueued events, then the apply process does not apply any user-enqueued events in its queue.

Removing the Rule Set for an Apply Process

You specify that you want to remove the rule set from an apply process by setting the `remove_rule_set` parameter to `true` in the `ALTER_APPLY` procedure in the `DBMS_APPLY_ADM` package. For example, the following procedure removes the rule set from an apply process named `strm01_apply`.

```
BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name      => 'strm01_apply',
    remove_rule_set => true);
END;
/
```

Note: If you remove a rule set for an apply process that applies captured events, then the apply process applies all captured events in its queue. Similarly, if you remove a rule set for an apply process that applies user-enqueued events, then the apply process applies all user-enqueued events in its queue.

Setting an Apply Process Parameter

You 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 4-34
- The `DBMS_APPLY_ADM.SET_PARAMETER` procedure in the *Oracle9i Supplied PL/SQL Packages and Types Reference* for detailed information about the apply process parameters
- ["Specifying Supplemental Logging at a Source Database"](#) on page 12-9

Setting the Apply User for an Apply Process

The apply user is the user who applies all DML statements and DDL statements and who runs user-defined apply handlers. You set the apply user for an apply process using the `apply_user` parameter in the `ALTER_APPLY` procedure in the `DBMS_APPLY_ADM` package. 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;
/
```

The user specified by the `apply_user` parameter must have the necessary privileges to perform DML and DDL changes on the apply objects and to run any apply handlers. The specified user must also have dequeue privileges on the queue used by the apply process and privileges to execute the rule set and transformation functions used by the apply process. 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
- [Chapter 19, "Streams Messaging Example"](#) for an example of creating a message handler

Setting the Message Handler for an Apply Process

You 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_proc` procedure in the `hr` schema.

```
BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name      => 'strm03_apply',
    message_handler => 'hr.mes_proc');
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 a DML Handler

This section contains instructions for creating, setting, and removing a DML handler.

See Also:

- ["Event Processing with an Apply Process"](#) on page 4-3
- [Chapter 20, "Single Database Capture and Apply Example"](#) for a detailed example that uses a DML handler

Creating a DML Handler

A DML handler must have the following signature:

```
PROCEDURE user_procedure (  
    parameter_name IN SYS.AnyData);
```

Here, *user_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 `SYS.AnyData` encapsulation of a row LCR.

The following restrictions apply to the user procedure:

- Do not execute `COMMIT` or `ROLLBACK` statements. Doing so may endanger the consistency of the transaction that contains the LCR.
- If you are manipulating a row using the `EXECUTE` member procedure for the row LCR, then do not attempt to manipulate more than one row in a row operation. You must construct and execute manually any DML statements that manipulate more than one row.
- If the command type is `UPDATE` or `DELETE`, then row operations resubmitted using the `EXECUTE` member procedure for the LCR must include the entire key in the list of old values. The key is the primary key, unless a substitute key has been specified by the `SET_KEY_COLUMNS` procedure.
- If the command type is `INSERT`, then row operations resubmitted using the `EXECUTE` member procedure for the LCR should include the entire key in the list of new values. Otherwise, duplicate rows are possible. The key is the primary key, unless a substitute key has been specified by the `SET_KEY_COLUMNS` procedure.

A DML handler can be used for any customized processing of row LCRs. For example, the handler may modify an LCR and then execute it using the `EXECUTE` member procedure for the LCR. When you execute a row LCR in a DML handler, the apply process applies the row LCR without calling any DML handler or error handler for the row LCR.

You may also use a DML handler for recording the history of DML changes. For example, a DML handler may insert information about an LCR it processes into a table and then apply the LCR using the `EXECUTE` member procedure. To create such a DML handler, first create a table to hold the history information:

```
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,
    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;
```

Then, create the procedure that inserts the information in the row LCR into the `history_row_lcrs` table and executes the row LCR:

```
CREATE OR REPLACE PROCEDURE history_dml(in_any IN SYS.ANYDATA)
IS
    lcr  SYS.LCR$_ROW_RECORD;
    rc   PLS_INTEGER;
BEGIN
    -- Access the LCR
    rc := in_any.GETOBJECT(lcr);
    -- Insert information in the LCR into the history_row_lcrs table
    INSERT INTO strmadmin.history_row_lcrs VALUES
        (SYSDATE, lcr.GET_SOURCE_DATABASE_NAME(), lcr.GET_COMMAND_TYPE(),
         lcr.GET_OBJECT_OWNER(), lcr.GET_OBJECT_NAME(),
         lcr.GET_TAG(), lcr.GET_TRANSACTION_ID(), lcr.GET_SCN(),
         lcr.GET_VALUES('old'), lcr.GET_VALUES('new', 'n'));
    -- Apply row LCR
    lcr.EXECUTE(true);
END;
/
```

Note: You must specify an unconditional supplemental log group at the source database for any columns needed by a DML handler at the destination database. This example DML handler does not require any additional supplemental logging because it simply records information about the row LCR and does not manipulate the row LCR in any other way.

See Also: ["Specifying Supplemental Logging at a Source Database"](#) on page 12-9

Setting a DML Handler

A DML handler processes each row LCR dequeued by any apply process that contains a specific operation on a specific table. You can specify multiple DML handlers on the same table, to handle different operations on the table. All apply processes that apply changes to the specified table in the local database use the specified DML handler.

You set the DML handler using the `SET_DML_HANDLER` procedure in the `DBMS_APPLY_ADM` package. For example, the following procedure sets the DML handler for `UPDATE` operations on the `hr.locations` table. Therefore, when any apply process that applies changes locally dequeues a row LCR containing an `UPDATE` operation on the `hr.locations` table, the apply process sends the row LCR to the `history_dml` PL/SQL procedure in the `strmadmin` schema for processing. The apply process does not apply a row LCR containing such a change directly.

```
BEGIN
  DBMS_APPLY_ADM.SET_DML_HANDLER(
    object_name      => 'hr.locations',
    object_type      => 'TABLE',
    operation_name   => 'UPDATE',
    error_handler    => false,
    user_procedure   => 'strmadmin.history_dml',
    apply_database_link => NULL);
END;
/
```

Note:

- If an apply process applies changes to a remote non-Oracle database, then it may use a different DML handler for the same table. You can run the `SET_DML_HANDLER` procedure in the `DBMS_APPLY_ADM` package to specify a DML handler for changes that will be applied to a remote non-Oracle database by setting the `apply_database_link` parameter to a non-NULL value.
 - When you run the `SET_DML_HANDLER` procedure, you specify the object for which the handler is used, and Oracle checks to ensure that the specified object exists in the local destination database. If the object does not exist, then an error is raised. Therefore, if name of the object is different at the source database and destination database, then use a rule-based transformation to convert the object name in the row LCR before the row LCR is applied.
-

See Also:

- ["Apply Process Configuration in an Oracle to Non-Oracle Environment"](#) on page 9-4
- ["Rule-Based Transformations"](#) on page 6-25

Removing a DML Handler

You remove a DML 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 removes the DML handler for `UPDATE` operations on the `hr.locations` table. After the DML handler is removed, any apply process that applies changes locally will apply a row LCR containing such a change directly.

```
BEGIN
  DBMS_APPLY_ADM.SET_DML_HANDLER(
    object_name   => 'hr.locations',
    object_type   => 'TABLE',
    operation_name => 'UPDATE',
    error_handler => false,
    user_procedure => NULL);
END;
/
```

Managing the DDL Handler for an Apply Process

This section contains instructions for creating, specifying, and removing the DDL handler for an apply process.

Note: 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.

See Also:

- ["Event Processing with an Apply Process"](#) on page 4-3
- *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about the `EXECUTE` member procedure for LCR types

Creating a DDL Handler for an Apply Process

A DDL handler must have the following signature:

```
PROCEDURE handler_procedure (  
    parameter_name IN SYS.AnyData);
```

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 `SYS.AnyData` encapsulation of a DDL LCR.

A DDL handler can be used for any customized processing of DDL LCRs. For example, the handler may modify the LCR and then execute it using the `EXECUTE` member procedure for the LCR. When you execute a DDL LCR in a DDL handler, the apply process applies the LCR without calling the DDL handler again.

You may also use a DDL handler to record the history of DDL changes. For example, a DDL handler may insert information about an LCR it processes into a table and then apply the LCR using the `EXECUTE` member procedure.

To create such a DDL handler, first create a table to hold the history information:

```
CREATE TABLE strmadmin.history_ddl_lcrs(  
    timestamp          DATE,  
    source_database_name VARCHAR2(128),  
    command_type       VARCHAR2(30),  
    object_owner       VARCHAR2(32),  
    object_name        VARCHAR2(32),  
    object_type        VARCHAR2(18),  
    ddl_text           CLOB,  
    logon_user         VARCHAR2(32),  
    current_schema     VARCHAR2(32),  
    base_table_owner   VARCHAR2(32),  
    base_table_name    VARCHAR2(32),  
    tag                RAW(10),  
    transaction_id     VARCHAR2(10),  
    scn                NUMBER);
```

Then, create the procedure that inserts the information in the DDL LCR into the `history_ddl_lcrs` table and executes the DDL LCR:

```
CREATE OR REPLACE procedure history_ddl(in_any IN SYS.ANYDATA)
IS
  lcr      SYS.LCR$_DDL_RECORD;
  rc      PLS_INTEGER;
  ddl_text CLOB;
BEGIN
  -- Access the LCR
  rc := in_any.GETOBJECT(lcr);
  DBMS_LOB.CREATETEMPORARY(ddl_text, TRUE);
  lcr.GET_DDL_TEXT(ddl_text);
  -- Insert DDL LCR information into history_ddl_lcrs table
  INSERT INTO strmadmin.history_ddl_lcrs VALUES(
    SYSDATE, lcr.GET_SOURCE_DATABASE_NAME(), lcr.GET_COMMAND_TYPE(),
    lcr.GET_OBJECT_OWNER(), lcr.GET_OBJECT_NAME(), lcr.GET_OBJECT_TYPE(),
    ddl_text, lcr.GET_LOGON_USER(), lcr.GET_CURRENT_SCHEMA(),
    lcr.GET_BASE_TABLE_OWNER(), lcr.GET_BASE_TABLE_NAME(), lcr.GET_TAG(),
    lcr.GET_TRANSACTION_ID(), lcr.GET_SCN());
  -- Apply DDL LCR
  lcr.EXECUTE();
  -- Free temporary LOB space
  DBMS_LOB.FREETEMPORARY(ddl_text);
END;
/
```

Setting the DDL Handler for an Apply Process

A DDL handler processes all DDL LCRs dequeued by an apply process. You set the DDL handler for an apply process using the `ddl_handler` parameter in the `ALTER_APPLY` procedure in the `DBMS_APPLY_ADM` package. For example, the following procedure sets the DDL handler for an apply process named `strm01_apply` to the `history_ddl` procedure in the `strmadmin` schema.

```
BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name => 'strm01_apply',
    ddl_handler => 'strmadmin.history_ddl');
END;
/
```

Removing the DDL Handler for an Apply Process

A DDL handler processes all DDL LCRs dequeued by an apply process. You remove the DDL handler for an apply process by setting the `remove_ddl_handler` parameter to `true` in the `ALTER_APPLY` procedure in the `DBMS_APPLY_ADM` package. For example, the following procedure removes the DDL handler from an apply process named `strm01_apply`.

```
BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name          => 'strm01_apply',
    remove_ddl_handler => true);
END;
/
```

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 ["Creating a DML Handler"](#) on page 14-14 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 an exception 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 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,
  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$_ROW_RECORD;
    ret             PLS_INTEGER;
    vc              VARCHAR2(30) ;
    errlog_rec      errorlog%ROWTYPE ;
    ov2            SYS.LCR$_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 ;
  ad := DBMS_STREAMS.GET_INFORMATION('SENDER');
  ret := ad.GETVARCHAR2(errlog_rec.sender);
  -- 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, 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: ["Setting the Tag Values Generated by the Current Session"](#) on page 16-26

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. All apply processes that apply changes to the specified table in the local database use the specified error handler.

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 an exception queue.

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);
END;
/
```

Removing an Error Handler

You remove 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 removes 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);
END;
/
```

Note: The `error_handler` parameter need not be specified.

Managing the Substitute Key Columns for a Table

This section contains instructions for setting and removing the substitute key columns for a table.

See Also: ["Substitute Key Columns"](#) on page 4-11

Setting Substitute Key Columns for a Table

When an apply process applies changes to a table, substitute key columns can either replace the primary key columns for a table that has a primary key or act as the primary key columns for a table that does not have a primary key. You set the substitute key columns for a table using the `SET_KEY_COLUMNS` procedure in the `DBMS_APPLY_ADM` package. This setting applies to all of the apply processes that apply local changes to the database.

For example, to set the substitute key columns for the `hr.employees` table to the `first_name`, `last_name`, and `hire_date` columns, replacing the `employee_id` column, run the following procedure:

```
BEGIN
  DBMS_APPLY_ADM.SET_KEY_COLUMNS(
    object_name      => 'hr.employees',
    column_list     => 'first_name,last_name,hire_date');
END;
/
```

Note:

- You must specify an unconditional supplemental log group at the source database for all of the columns specified as substitute key columns in the `column_list` or `column_table` parameter at the destination database. In this example, you would specify an unconditional supplemental log group including the `first_name`, `last_name`, and `hire_date` columns in the `hr.employees` table.
 - If an apply process applies changes to a remote non-Oracle database, then it may use different substitute key columns for the same table. You can run the `SET_KEY_COLUMNS` procedure in the `DBMS_APPLY_ADM` package to specify substitute key columns for changes that will be applied to a remote non-Oracle database by setting the `apply_database_link` parameter to a non-NULL value.
-
-

See Also:

- ["Specifying Supplemental Logging at a Source Database"](#) on page 12-9
- ["Apply Process Configuration in an Oracle to Non-Oracle Environment"](#) on page 9-4

Removing the Substitute Key Columns for a Table

You remove the substitute key columns for a table by specifying `NULL` for the `column_list` or `column_table` parameter in the `SET_KEY_COLUMNS` procedure in the `DBMS_APPLY_ADM` package. If the table has a primary key, then the table's primary key is used by any apply process for local changes to the database after you remove the substitute primary key.

For example, to remove the substitute key columns for the `hr.employees` table, run the following procedure:

```
BEGIN
  DBMS_APPLY_ADM.SET_KEY_COLUMNS(
    object_name => 'hr.employees',
    column_list => NULL);
END;
/
```

Managing Streams Conflict Resolution

This section contains instructions for creating, specifying, and removing update conflict handlers a table. All apply processes running on a database that apply changes to the specified table locally use the specified update conflict handler.

See Also: [Chapter 7, "Streams Conflict Resolution"](#)

Setting an Update Conflict Handler

You set an update conflict handler using the `SET_UPDATE_CONFLICT_HANDLER` procedure in the `DBMS_APPLY_ADM` package. You can use one of the following prebuilt methods when you create an update conflict resolution handler:

- `OVERWRITE`
- `DISCARD`
- `MAXIMUM`
- `MINIMUM`

For example, suppose a Streams environment captures changes to the `hr.jobs` table at `dbs1.net` and propagates these changes to the `dbs2.net` destination database, where they are applied. In this environment, applications can perform DML changes on the `hr.jobs` table at both databases, but, if there is a conflict for a particular DML change, then the change at the `dbs1.net` database should always overwrite the change at the `dbs2.net` database. In this environment, you can accomplish this goal by specifying an `OVERWRITE` handler at the `dbs2.net` database.

To specify an update conflict handler for the `hr.jobs` table in the `hr` schema at the `dbs2.net` database, run the following procedure at `dbs2.net`:

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

Note:

- The `resolution_column` is not used for `OVERWRITE` and `DISCARD` methods, but one of the columns in the `column_list` still must be specified.
 - You must specify a conditional supplemental log group at the source database for all of the columns in the `column_list` at the destination database. In this example, you would specify a conditional supplemental log group including the `job_title`, `min_salary`, and `max_salary` columns in the `hr.jobs` table at the `dbssl.net` database.
 - Conflict resolution does not support LOB columns. Therefore, you should not include LOB columns in the `column_list` parameter when running `SET_UPDATE_CONFLICT_HANDLER`.
-
-

See Also:

- ["Specifying Supplemental Logging at a Source Database"](#) on page 12-9
- [Chapter 23, "Multiple Source Replication Example"](#) for an example Streams environment that illustrates using the `MAXIMUM` prebuilt method for time-based conflict resolution

Modifying an Existing Update Conflict Handler

You can modify an existing update conflict handler by running the `SET_UPDATE_CONFLICT_HANDLER` procedure in the `DBMS_APPLY_ADM` package. To update an existing conflict handler, specify the same table and resolution column as the existing conflict handler.

To modify the update conflict handler created in ["Setting an Update Conflict Handler"](#) on page 14-30, you specify the `hr.jobs` table and the `job_title` column as the resolution column. You can modify this update conflict handler by specifying a different type of prebuilt method or a different column list, or both. However, if you want to change the resolution column for an update conflict handler, then you must remove and re-create the handler.

For example, suppose the environment changes, and you want changes from `dbssl.net` to be discarded in the event of a conflict, whereas previously changes

from `dbs1.net` overwrote changes at `dbs2.net`. You can accomplish this goal by specifying a `DISCARD` handler at the `dbs2.net` database.

To modify the existing update conflict handler for the `hr.jobs` table in the `hr` schema at the `dbs2.net` database, run the following procedure:

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

Removing an Existing Update Conflict Handler

You can remove an existing update conflict handler by running the `SET_UPDATE_CONFLICT_HANDLER` procedure in the `DBMS_APPLY_ADM` package. To remove an existing conflict handler, specify `NULL` for the method, and specify the same table, column list, and resolution column as the existing conflict handler.

For example, suppose you want to remove the update conflict handler created in ["Setting an Update Conflict Handler"](#) on page 14-30 and then modified in ["Modifying an Existing Update Conflict Handler"](#) on page 14-31. To remove this update conflict handler, run the following procedure:

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


Managing Apply Errors

This section contains instructions for retrying and deleting apply errors.

See Also:

- ["Exception Queues"](#) on page 4-37
- ["Checking for Apply Errors"](#) on page 17-36
- ["Displaying Detailed Information About Apply Errors"](#) on page 17-37
- ["Considerations for Applying DML Changes to Tables"](#) on page 4-10 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: ["Setting the Tag Values Generated by the Current Session"](#) on page 16-26

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 its exception 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.

Setting Instantiation SCNs at a Destination Database

An instantiation SCN instructs an apply process at a destination database to apply changes to a database object that committed after a specific SCN at a source database. You can set instantiation SCNs in one of the following ways:

- Perform instantiation of the relevant database objects by exporting them at the source database and importing them into the destination database. In this case, the instantiation creates the database objects at the destination database, populates them with the data from the source database, and sets the relevant instantiation SCNs.
- Perform a metadata only export/import by setting the `ROWS` parameter to `n` during export at the source database or import at the destination database, or both. In this case, the database objects are instantiated, but no data is imported.
- Set the instantiation SCN using the `SET_TABLE_INSTANTIATION_SCN`, `SET_SCHEMA_INSTANATIATION_SCN`, and `SET_GLOBAL_INSTANTIATION_SCN` procedures in the `DBMS_APPLY_ADM` package.

See Also:

- ["Instantiation"](#) on page 2-13
- ["Instantiation SCN and Ignore SCN"](#) on page 4-27

Setting Instantiation SCNs Using Export/Import

This section discusses setting instantiation SCNs by performing an export/import. The information in this section applies to both metadata export/import operations and to export/import operations that import rows.

To set instantiation SCNs for database objects using Export/Import, first export them at the source database with the `OBJECT_CONSISTENT` export parameter set to `Y`, or use a more stringent degree of consistency. Then, import them at the destination database with the `STREAMS_INSTANTIATION` import parameter set to `Y`.

Note:

- If a non-NULL instantiation SCN already exists for a database object at a destination database that performs an import, then the import does not update the instantiation SCN for that database object.
 - During an export for a Streams instantiation, make sure no DDL changes are made to objects being exported.
 - Any table supplemental logging specifications for the tables exported from the export database are retained when the tables are imported at the import database.
-
-

The following sections describe the instantiation SCNs set for different types of export/import operations. These sections refer to prepared tables. Prepared tables are tables that have been prepared for instantiation using the `PREPARE_TABLE_INSTANTIATION`, `PREPARE_SCHEMA_INSTANTIATION`, or `PREPARE_GLOBAL_INSTANTIATION` procedures in the `DBMS_CAPTURE_ADM` package. A table must be a prepared table before export in order for an instantiation SCN to be set for it during import. However, the database and schemas do not need to be prepared before the export in order for their instantiation SCNs to be set during import.

Full Database Export and Full Database Import

A full database export and full database import sets the following instantiation SCNs at the import database:

- The database, or global, instantiation SCN
- The schema instantiation SCN for each imported user
- The table instantiation SCNs for each prepared table that is imported

Full Database or User Export and User Import

A full database or user export and user import sets the following instantiation SCNs at the import database:

- The schema instantiation SCN for each imported user
- The table instantiation SCN for each prepared tables that is imported

Full Database, User, or Table Export and Table Import

Any export that includes one or more tables and a table import sets the table instantiation SCN for each prepared table that is imported at the import database.

See Also:

- ["Setting Export and Import Parameters Relevant to Streams"](#) on page 11-8 and *Oracle9i Database Utilities* for information about using Export/Import
- ["Configuring a Capture-Based Streams Environment"](#) on page 11-15 for more information about performing export/import operations to set instantiation SCNs when configuring a Streams environment
- ["Preparing Database Objects for Instantiation at a Source Database"](#) on page 12-11

Setting Instantiation SCNs Using the DBMS_APPLY_ADM Package

You can set an instantiation SCN at a destination database for a specified table, a specified schema, or an entire database using one of the following procedures in the DBMS_APPLY_ADM package:

- SET_TABLE_INSTANTIATION_SCN
- SET_SCHEMA_INSTANTIATION_SCN
- SET_GLOBAL_INSTANTIATION_SCN

If you set the instantiation SCN for a schema using SET_SCHEMA_INSTANTIATION_SCN, then you should set the instantiation SCN for each table in the schema using SET_TABLE_INSTANTIATION_SCN. Similarly, if you set the instantiation SCN for a database using SET_GLOBAL_INSTANTIATION_SCN, then you should set the instantiation SCN for each schema in the database using SET_SCHEMA_INSTANTIATION_SCN

[Table 14-1](#) lists each procedure and the types of statements for which they set an instantiation SCN.

Table 14–1 Set Instantiation SCN Procedures and the Statements They Cover

Procedure	Sets Instantiation SCN for	Examples
SET_TABLE_INSTANTIATION_SCN	DML and DDL statements on tables, except CREATE TABLE DDL statements on table indexes and table triggers	UPDATE ALTER TABLE DROP TABLE CREATE, ALTER, or DROP INDEX on a table CREATE, ALTER, or DROP TRIGGER on a table
SET_SCHEMA_INSTANTIATION_SCN	DDL statements on users, except CREATE USER DDL statements on all database objects that have a non-PUBLIC owner, except for those DDL statements handled by a table-level instantiation SCN	CREATE TABLE ALTER USER DROP USER CREATE PROCEDURE
SET_GLOBAL_INSTANTIATION_SCN	DDL statements on database objects other than users with no owner DDL statements on database objects owned by public CREATE USER statements	CREATE USER CREATE TABLESPACE

The following example sets the instantiation SCN for the `hr.departments` table at the `hrdb2.net` database to the current SCN by running the following procedure at the source database `hrdb1.net`:

```

DECLARE
    iscn NUMBER;          -- Variable to hold instantiation SCN value
BEGIN
    iscn := DEMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER();
    DBMS_APPLY_ADM.SET_TABLE_INSTANTIATION_SCN@HRDB2.NET(
        source_object_name => 'hr.departments',
        source_database_name => 'hrdb1.net',
        instantiation_scn => iscn);
END;
/

```

Note:

- If a relevant instantiation SCN is not present, then an error is raised during apply.
 - The `SET_SCHEMA_INSTANTIATION_SCN` procedure does not set the instantiation SCN for any of the tables in the schema.
 - The `SET_GLOBAL_INSTANTIATION_SCN` procedure does not set the instantiation SCN for any of the schemas in the database.
 - If an apply process applies changes to a remote non-Oracle database, then set the `apply_database_link` parameter to the database link used for remote apply when you set the instantiation SCN.
-
-

See Also:

- [Chapter 11, "Configuring a Streams Environment"](#) for more information when to set instantiation SCNs when you are configuring a Streams environment
- [Chapter 22, "Single Source Heterogeneous Replication Example"](#) for a detailed example that uses the `SET_TABLE_INSTANTIATION_SCN` procedure
- The information about the `DBMS_APPLY_ADM` package in the *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about which instantiation SCN can be used for a DDL LCR
- ["Instantiation in an Oracle to Non-Oracle Environment"](#) on page 9-7

Managing Rules and Rule-Based Transformations

A Streams environment uses rules to control the behavior of capture processes, propagations, and apply processes. 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, or apply of an event. This chapter contains instructions for managing rule sets, rules, and rule-based transformations.

This chapter contains these topics:

- [Managing Rule Sets and Rules](#)
- [Managing Privileges on Evaluation Contexts, Rule Sets, and Rules](#)
- [Managing Rule-Based Transformations](#)

Each task described in this section should be completed by a Streams administrator that has been granted the appropriate privileges, unless specified otherwise.

See Also:

- [Chapter 5, "Rules"](#)
- [Chapter 6, "How Rules Are Used In Streams"](#)
- ["Rule-Based Transformations" on page 6-25](#)
- [Chapter 24, "Rule-Based Application Example"](#)
- ["Configuring a Streams Administrator" on page 11-2](#)

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 12-14](#)
- ["Disabling a Propagation Job" on page 13-17](#)
- ["Stopping an Apply Process" on page 14-7](#)

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 can also use the following procedures in the `DBMS_STREAMS_ADM` package to create a rule set automatically, if one does not exist for a Streams capture process, propagation, or apply process:

- `ADD_GLOBAL_PROPAGATION_RULES`
- `ADD_GLOBAL_RULES`
- `ADD_SCHEMA_PROPAGATION_RULES`
- `ADD_SCHEMA_RULES`
- `ADD_SUBSET_RULES`
- `ADD_TABLE_PROPAGATION_RULES`
- `ADD_TABLE_RULES`

See Also:

- ["Example of Creating a Capture Process Using DBMS_STREAMS_ADM"](#) on page 12-3
- ["Example of Creating a Propagation Using DBMS_STREAMS_ADM"](#) on page 13-9
- ["Example of Creating an Apply Process Using DBMS_STREAMS_ADM"](#) on page 14-3

Creating a Rule

The following is an example that runs the `CREATE_RULE` procedure in the `DBMS_RULE_ADM` package to create a rule:

```
BEGIN
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name      => 'strmadmin.hr_dml',
    condition      => ':dml.get_object_owner() = ''HR'' ',
    evaluation_context => NULL);
END;
/
```

Running this procedure performs the following actions:

- Creates a rule named `hr_dml` in the `strmadmin` schema. A rule with the same name and owner must not exist.
- Creates a condition that evaluates to `TRUE` for any DML change to a table in the `hr` schema

In this example, no evaluation context is specified for the rule. Therefore, the rule will either inherit the evaluation context of any rule set to which it is added, or it will be assigned an evaluation context explicitly when the `DBMS_RULE_ADM.ADD_RULE` procedure is run to add it to a rule set. At this point, the rule cannot be evaluated because it is not part of any rule set.

You can also use the following procedures in the `DBMS_STREAMS_ADM` package to create rules and add them to a rule set automatically:

- `ADD_GLOBAL_PROPAGATION_RULES`
- `ADD_GLOBAL_RULES`
- `ADD_SCHEMA_PROPAGATION_RULES`
- `ADD_SCHEMA_RULES`
- `ADD_SUBSET_RULES`
- `ADD_TABLE_PROPAGATION_RULES`
- `ADD_TABLE_RULES`

See Also:

- ["Example of Creating a Capture Process Using DBMS_STREAMS_ADM"](#) on page 12-3
- ["Example of Creating a Propagation Using DBMS_STREAMS_ADM"](#) on page 13-9
- ["Example of Creating an Apply Process Using DBMS_STREAMS_ADM"](#) on page 14-3

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
- Change a rule's action context
- Remove a rule's action context
- Change the comment for a rule
- Remove the comment for a rule

For example, suppose you want to change the condition of the rule created in ["Creating a Rule"](#) on page 15-3. 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 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 modify the rule's action context to add the rule-based transformation.

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_TABLE_RULES`, `DBA_STREAMS_SCHEMA_RULES`, or `DBA_STREAMS_GLOBAL_RULES` data dictionary view.
2. Use the copied rule condition to create a new rule by modifying the condition.
3. Add the new rule to the rule set for the Streams capture process, propagation, or apply process.
4. 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 6-25
- [Chapter 17, "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 5-15
- The `GRANT_SYSTEM_PRIVILEGE` and `GRANT_OBJECT_PRIVILEGE` procedures in the `DBMS_RULE_ADM` package in *Oracle9i Supplied 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 `strmadmin` 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   => 'strmadmin',
    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 `strmadmin` 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 `strmadmin` 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 `strmadmin` 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      => 'strmadmin');
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 evaluates to `TRUE`. You use a rule action context to specify a rule-based transformation. In the name-value pair that specifies a rule-based transformation in an action context, the name is `STREAMS$_TRANSFORM_FUNCTION` and the value is a `SYS.AnyData` instance containing the name of the PL/SQL function that performs the 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](#)

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 table name in the DDL text should be changed in the same way.
-
-

See Also: ["Rule-Based Transformations"](#) on page 6-25

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 LCR, and the function must return a `SYS.AnyData` encapsulation of an LCR.

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 does not raise any exceptions. Exceptions may cause the 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.

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');
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. You must use caution when altering an action context with more than one name value pair, as described in ["Altering a Rule-Based Transformation"](#) on page 15-18.

This example creates subset rules for an apply process on a database named `dbsl.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     => 'dbsl.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. The ["Flexible Configuration for Sharing Data from a Single Database"](#) on page 22-40 contains an example of this.
-
-

4. Determine the names of the system-created rules by running the following query:

```
SELECT RULE_NAME, SUBSETTING_OPERATION FROM DBA_STREAMS_TABLE_RULES
       WHERE TABLE_NAME='DEPARTMENTS' AND DML_CONDITION='location_id=1700';
```

This query displays output similar to the following:

RULE_NAME	SUBSET
DEPARTMENTS5	INSERT
DEPARTMENTS6	UPDATE
DEPARTMENTS7	DELETE

Note: You can also obtain this information using the `OUT` parameters when you run `ADD_SUBSET_RULES`.

Because these are subset rules, two of them contain a non-NULL action context that performs an internal transformation:

- The rule with a subsetting condition of `INSERT` contains an internal transformation that converts updates into inserts if the update changes the value of the `location_id` column to 1700 from some other value. The internal transformation does not affect inserts.
- The rule with a subsetting condition of `DELETE` contains an internal transformation that converts updates into deletes if the update changes the value of the `location_id` column from 1700 to a different value. The internal transformation does not affect deletes.

In this example, you can confirm that the rules `DEPARTMENTS5` and `DEPARTMENTS7` have a non-NULL action context, and that the rule `DEPARTMENTS6` has a NULL action context, by running the following query:

```
COLUMN RULE_NAME HEADING 'Rule Name' FORMAT A13
COLUMN ACTION_CONTEXT_NAME HEADING 'Action Context Name' FORMAT A27
COLUMN ACTION_CONTEXT_VALUE HEADING 'Action Context Value' FORMAT 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.

- Alter the action context of each subset rule to add the name-value pair for the rule-based transformation. Make sure no other users are modifying the action context at the same time. The name in the name-value pair must be STREAMS\$_TRANSFORM_FUNCTION.

Add the rule-based transformation to the DEPARTMENTS5 rule. The following statement preserves the existing name-value pairs in the action context by selecting the action context into a variable before adding the new pair.

```

DECLARE
    action_ctx      SYS.RE$NV_LIST;
    ac_name         VARCHAR2(30) := 'STREAMS$_TRANSFORM_FUNCTION';
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='DEPARTMENTS5';
    action_ctx.ADD_PAIR(ac_name,
                       SYS.ANYDATA.CONVERTVARCHAR2('hr.executive_to_management'));
    DBMS_RULE_ADM.ALTER_RULE(
        rule_name      => 'strmadmin.departments5',
        action_context => action_ctx);
END;
/

```


Add the rule-based transformation to the DEPARTMENTS6 rule. This statement does not need to query for the action context because it is NULL for the DEPARTMENTS6 rule.

```

DECLARE
    action_ctx      SYS.RE$NV_LIST;
    ac_name         VARCHAR2(30) := 'STREAMS$_TRANSFORM_FUNCTION';
BEGIN
    action_ctx := SYS.RE$NV_LIST(SYS.RE$NV_ARRAY());
    action_ctx.ADD_PAIR(ac_name,
                       SYS.ANYDATA.CONVERTVARCHAR2('hr.executive_to_management'));
    DBMS_RULE_ADM.ALTER_RULE(
        rule_name     => 'strmadmin.departments6',
        action_context => action_ctx);
END;
/
    
```

Add the rule-based transformation to the DEPARTMENTS7 rule. This statements queries for the existing action context and inserts it into a variable before adding a new name-value pair.

```

DECLARE
    action_ctx      SYS.RE$NV_LIST;
    ac_name         VARCHAR2(30) := 'STREAMS$_TRANSFORM_FUNCTION';
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='DEPARTMENTS7';
    action_ctx.ADD_PAIR(ac_name,
                       SYS.ANYDATA.CONVERTVARCHAR2('hr.executive_to_management'));
    DBMS_RULE_ADM.ALTER_RULE(
        rule_name     => 'strmadmin.departments7',
        action_context => action_ctx);
END;
/
    
```

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

See Also: *Oracle9i Supplied 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 edit the action context to run a different transformation function. This example edits the action context to run a different function. If you edit the function itself, then you do not need to alter the action context.

This example alters a rule-based transformation for rule DEPARTMENTS5 by first removing the name-value pair with the name STREAMS\$_TRANSFORM_FUNCTION from the rule's action context and then adding a different name-value pair back to the rule's action context. This rule based transformation was added to the DEPARTMENTS5 rule in the example in ["Creating a Rule-Based Transformation"](#) on page 15-11.

If an action context contains name-value pairs in addition to the name-value pair that specifies the transformation, then be cautious when you modify the action context so that you do not change or remove any name-value pairs that are unrelated to the transformation.

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. If you specify a new transformation or alter an existing transformation for a subset rule, then make sure you preserve the name-value pairs that perform row migrations.

See Also: ["Row Migration"](#) on page 4-12

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. For the DEPARTMENTS5 rule, the transformation function is `executive_to_management`. To alter the transformation function, this step first removes the name-value pair containing the function name from the action context for the DEPARTMENTS5 rule. Then, this step adds a name-value pair containing the new function name to the rule's action context. 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.

To preserve any existing name-value pairs in the rule's action context, this example selects for the rule's action context into a variable before altering it:

```

DECLARE
    action_ctx          SYS.RE$NV_LIST;
    ac_name             VARCHAR2(30) := 'STREAMS$_TRANSFORM_FUNCTION';
BEGIN
    SELECT RULE_ACTION_CONTEXT
           INTO action_ctx
           FROM DBA_RULES R
           WHERE RULE_OWNER='STRMADMIN' AND RULE_NAME='DEPARTMENTS5';
    action_ctx.REMOVE_PAIR(ac_name);
    action_ctx.ADD_PAIR(ac_name,
                       SYS.ANYDATA.CONVERTVARCHAR2('hr.executive_to_lead'));
    DBMS_RULE_ADM.ALTER_RULE(
        rule_name      => 'strmadmin.departments5',
        action_context => action_ctx);
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.

Removing a Rule-Based Transformation

To remove a rule-based transformation from a rule, you remove the name-value pair with the name `STREAMS$_TRANSFORM_FUNCTION` from the rule's action. 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 15-11.

Removing a rule-based transformation means altering the action context of a rule. If an action context contains name-value pairs in addition to the name-value pair that specifies the transformation, then be cautious when you modify the action context so that you do not change or remove any name-value pairs that are unrelated to the transformation.

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. If you specify a new transformation or alter an existing transformation for a subset rule, then make sure you preserve the name-value pairs that perform row migrations.

This example queries for the rule's action context and places it in a variable before removing the name-value pair for the rule-based transformation:

```
DECLARE
  action_ctx      SYS.RE$NV_LIST;
  ac_name         VARCHAR2(30) := 'STREAMS$_TRANSFORM_FUNCTION';
BEGIN
  SELECT RULE_ACTION_CONTEXT
  INTO action_ctx
  FROM DBA_RULES R
  WHERE RULE_OWNER='STRMADMIN' AND RULE_NAME='DEPARTMENTS5';
  action_ctx.REMOVE_PAIR(ac_name);
  DBMS_RULE_ADM.ALTER_RULE(
    rule_name      => 'strmadmin.departments5',
    action_context => action_ctx);
END;
/
```

To ensure that the transformation function was removed, you can run the query in Step 1 on page 15-19. 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"](#) on page 4-12

Other Streams Management Tasks

This chapter provides instructions for managing logical change records (LCRs) and Streams tags, as well as instructions for performing a full database export/import in a Streams environment.

This chapter contains these topics:

- [Managing Logical Change Records \(LCRs\)](#)
- [Managing Streams Tags](#)
- [Performing Database Point-in-Time Recovery on a Destination Database](#)
- [Performing Full Database Export/Import on a Database Using Streams](#)

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 11-2

Managing Logical Change Records (LCRs)

This section describes managing logical change records (LCRs). Make sure you meet the following requirements when you create or modify an LCR:

- If you create or modify a row LCR, then make sure the `command_type` attribute is consistent with the presence or absence of old column values and the presence or absence of new column values.
- If you create or modify a DDL LCR, then make sure the `ddl_text` is consistent with the `base_table_name`, `base_table_owner`, `object_type`, `object_owner`, `object_name`, and `command_type` attributes.

Constructing and Enqueuing LCRs

Use the following LCR constructors to create LCRs:

- To create a row LCR that contains a change to a row that resulted from a data manipulation language (DML) statement, use the `SYS.LCR$_ROW_RECORD` constructor.
- To create a DDL LCR that contains a data definition language change, use the `SYS.LCR$_DDL_RECORD` constructor. Make sure the DDL text specified in the `ddl_text` attribute of each DDL LCR conforms to Oracle SQL syntax.

The following example creates a queue in an Oracle database and an apply process associated with the queue. Then, it creates a PL/SQL procedure that constructs a row LCR based on information passed to it and enqueues the row LCR into the queue:

1. Create a Streams queue in an Oracle database. This example assumes that the Streams administrator is `strmadmin` user.

```
CONNECT strmadmin/strmadminpw

BEGIN DEMS_STREAMS_ADM.SET_UP_QUEUE(
    queue_table      => 'strm04_queue_table',
    storage_clause   => NULL,
    queue_name       => 'strm04_queue');
END;
/
```


2. Create an apply process at the Oracle database to receive messages in the queue. Make sure the `apply_captured` parameter is set to `false` when you create the apply process, because the apply process will be applying user-enqueued events, not events captured by a capture process. Also, make sure the `apply_user` parameter is set to `hr`, because changes will be applied in to the `hr.regions` table, and the apply user must have privileges to make DML changes to this table.

```
BEGIN
  DBMS_APPLY_ADM.CREATE_APPLY(
    queue_name      => 'strm04_queue',
    apply_name      => 'strm04_apply',
    apply_captured  => false,
    apply_user      => 'hr');
END;
/
```

3. Create a rule set for the apply process and add a rule that applies DML changes to the `hr.regions` table made at the `dbssl.net` source database.

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name      => 'hr.regions',
    streams_type    => 'apply',
    streams_name    => 'strm04_apply',
    queue_name      => 'strm04_queue',
    include_dml     => true,
    include_ddl     => false,
    include_tagged_lcr => false,
    source_database => 'dbssl.net');
END;
/
```

4. Set the `disable_on_error` parameter for the apply process to `n`.

```
BEGIN
  DBMS_APPLY_ADM.SET_PARAMETER(
    apply_name      => 'strm04_apply',
    parameter       => 'disable_on_error',
    value           => 'n');
END;
/
```

5. Start the apply process.

```
EXEC DBMS_APPLY_ADM.START_APPLY('strm04_apply');
```

6. Create a procedure called `construct_row_lcr` that constructs a row LCR and then enqueues it into the queue created in Step 1.

```
CREATE OR REPLACE PROCEDURE construct_row_lcr(
    source_dbname  VARCHAR2,
    cmd_type       VARCHAR2,
    obj_owner      VARCHAR2,
    obj_name       VARCHAR2,
    old_vals       SYS.LCR$_ROW_LIST,
    new_vals       SYS.LCR$_ROW_LIST) AS
    eopt           DBMS_AQ.ENQUEUE_OPTIONS_T;
    mprop          DBMS_AQ.MESSAGE_PROPERTIES_T;
    enq_msgid      RAW(16);
    row_lcr        SYS.LCR$_ROW_RECORD;
BEGIN
    mprop.SENDER_ID := SYS.AQ$_AGENT('strmadmin', NULL, NULL);
    -- Construct the LCR based on information passed to procedure
    row_lcr := SYS.LCR$_ROW_RECORD.CONSTRUCT(
        source_database_name => source_dbname,
        command_type         => cmd_type,
        object_owner         => obj_owner,
        object_name          => obj_name,
        old_values            => old_vals,
        new_values            => new_vals);
    -- Enqueue the created row LCR
    DBMS_AQ.ENQUEUE(
        queue_name           => 'strm04_queue',
        enqueue_options      => eopt,
        message_properties   => mprop,
        payload               => SYS.AnyData.ConvertObject(row_lcr),
        msgid                 => enq_msgid);
END construct_row_lcr;
/
```

Note: The application does not need to specify a transaction identifier or SCN when it creates an LCR because the apply process generates these values and stores them in memory. If a transaction identifier or SCN is specified in the LCR, then the apply process ignores it and assigns a new value.

See Also: *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about LCR constructors

7. Create and enqueue LCRs using the `construct_row_lcr` procedure created in Step 2.
 - a. Create a row LCR that inserts a row into the `hr.regions` table.

```
CONNECT strmadmin/strmadminpw

DECLARE
  newunit1 SYS.LCR$_ROW_UNIT;
  newunit2 SYS.LCR$_ROW_UNIT;
  newvals  SYS.LCR$_ROW_LIST;
BEGIN
  newunit1 := SYS.LCR$_ROW_UNIT(
    'region_id',
    SYS.AnyData.ConvertNumber(5),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  newunit2 := SYS.LCR$_ROW_UNIT(
    'region_name',
    SYS.AnyData.ConvertVarchar2('Moon'),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  newvals := SYS.LCR$_ROW_LIST(newunit1,newunit2);
  construct_row_lcr(
    source_dbname => 'dbs1.net',
    cmd_type      => 'INSERT',
    obj_owner     => 'hr',
    obj_name      => 'regions',
    old_vals      => NULL,
    new_vals      => newvals);
END;
/
COMMIT;
```

You can connect as the `hr` user and query the `hr.regions` table to view the applied row change:

```
CONNECT hr/hr

SELECT * FROM hr.regions;
```

The row with a `region_id` of 5 should have Moon for the `region_name`.

- b. Create a row LCR that updates a row from the `hr.regions` table.

```
CONNECT strmadmin/strmadminpw

DECLARE
  oldunit1 SYS.LCR$_ROW_UNIT;
  oldunit2 SYS.LCR$_ROW_UNIT;
  oldvals  SYS.LCR$_ROW_LIST;
  newunit1 SYS.LCR$_ROW_UNIT;
  newvals  SYS.LCR$_ROW_LIST;
BEGIN
  oldunit1 := SYS.LCR$_ROW_UNIT(
    'region_id',
    SYS.AnyData.ConvertNumber(5),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  oldunit2 := SYS.LCR$_ROW_UNIT(
    'region_name',
    SYS.AnyData.ConvertVarchar2('Moon'),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  oldvals := SYS.LCR$_ROW_LIST(oldunit1,oldunit2);
  newunit1 := SYS.LCR$_ROW_UNIT(
    'region_name',
    SYS.AnyData.ConvertVarchar2('Mars'),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  newvals := SYS.LCR$_ROW_LIST(newunit1);
  construct_row_lcr(
    source_dbname => 'dbs1.net',
    cmd_type      => 'UPDATE',
    obj_owner     => 'hr',
    obj_name      => 'regions',
    old_vals      => oldvals,
    new_vals      => newvals);
END;
/
COMMIT;
```

You can connect as the `hr` user and query the `hr.regions` table to view the applied row change:

```
CONNECT hr/hr

SELECT * FROM hr.regions;
```

The row with a `region_id` of 5 should have Mars for the `region_name`.

c. Create a row LCR that deletes a row from the `hr.regions` table.

```
CONNECT strmadmin/strmadminpw

DECLARE
  oldunit1 SYS.LCR$_ROW_UNIT;
  oldunit2 SYS.LCR$_ROW_UNIT;
  oldvals  SYS.LCR$_ROW_LIST;
BEGIN
  oldunit1 := SYS.LCR$_ROW_UNIT(
    'region_id',
    SYS.AnyData.ConvertNumber(5),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  oldunit2 := SYS.LCR$_ROW_UNIT(
    'region_name',
    SYS.AnyData.ConvertVarchar2('Mars'),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  oldvals := SYS.LCR$_ROW_LIST(oldunit1,oldunit2);
  construct_row_lcr(
    source_dbname => 'dbs1.net',
    cmd_type      => 'DELETE',
    obj_owner     => 'hr',
    obj_name      => 'regions',
    old_vals      => oldvals,
    new_vals      => NULL);
END;
/
COMMIT;
```

You can connect as the `hr` user and query the `hr.regions` table to view the applied row change:

```
CONNECT hr/hr

SELECT * FROM hr.regions;
```

The row with a `region_id` of 5 should have been deleted.

The `use_old` Parameter in Some Row LCR Member Functions

Release 9.2.0.2 introduces a new parameter, `use_old`, in the following member functions for the `SYS.LCR$_ROW_RECORD` type:

- [GET_LOB_INFORMATION Member Function](#)
- [GET_VALUE Member Function](#)
- [GET_VALUES Member Function](#)

Currently, the `use_old` parameter is not documented in the *Oracle9i Supplied PL/SQL Packages and Types Reference*. The following sections replace the sections for these member functions in the *Oracle9i Supplied PL/SQL Packages and Types Reference*.

GET_LOB_INFORMATION Member Function

Gets the LOB information for the column.

The return value can be one of the following:

<code>DBMS_LCR.NOT_A_LOB</code>	<code>CONSTANT NUMBER := 1;</code>
<code>DBMS_LCR.NULL_LOB</code>	<code>CONSTANT NUMBER := 2;</code>
<code>DBMS_LCR.INLINE_LOB</code>	<code>CONSTANT NUMBER := 3;</code>
<code>DBMS_LCR.EMPTY_LOB</code>	<code>CONSTANT NUMBER := 4;</code>
<code>DBMS_LCR.LOB_CHUNK</code>	<code>CONSTANT NUMBER := 5;</code>
<code>DBMS_LCR.LAST_LOB_CHUNK</code>	<code>CONSTANT NUMBER := 6;</code>

Returns `NULL` if the specified column does not exist.

If the command type of the row LCR is `UPDATE`, then specifying `'Y'` for the `use_old` parameter is a convenient way to get the value of the columns.

Syntax

```
MEMBER FUNCTION GET_LOB_INFORMATION(
    value_type    IN VARCHAR2,
    column_name   IN VARCHAR2,
    use_old       IN VARCHAR2    DEFAULT 'Y')
RETURN NUMBER;
```

Parameters**Table 16–1** *GET_LOB_INFORMATION Function Parameters*

Parameter	Description
value_type	The type of value to return for the column, either old or new
column_name	The name of the column
use_old	If Y and value_type is new, and no new value exists, then returns the corresponding old value. If N and value_type is new, then does not return the old value if no new value exists. If value_type is old or if the command_type of the row LCR is not UPDATE, then the value of the use_old parameter is ignored. NULL is not a valid specification for the use_old parameter.

GET_VALUE Member Function

Returns the old or new value for the specified column, depending on the value type specified.

If the command type of the row LCR is UPDATE, then specifying 'Y' for the use_old parameter is a convenient way to get the value of a column.

Syntax

```
MEMBER FUNCTION GET_VALUE(
    value_type    IN VARCHAR2,
    column_name   IN VARCHAR2,
    use_old       IN VARCHAR2    DEFAULT 'Y')
RETURN SYS.AnyData;
```

Parameters

Table 16–2 GET_VALUE Procedure Parameters

Parameter	Description
<code>value_type</code>	The type of value to return for the column. Specify <code>old</code> to get the old value for the column. Specify <code>new</code> to get the new value for the column.
<code>column_name</code>	The column name. If the column is present and has a <code>NULL</code> value, returns a <code>SYS.AnyData</code> instance containing a <code>NULL</code> value. If the column value is absent, returns a <code>NULL</code> .
<code>use_old</code>	<p>If <code>Y</code> and <code>value_type</code> is <code>new</code>, and no new value exists, then returns the corresponding old value.</p> <p>If <code>N</code> and <code>value_type</code> is <code>new</code>, then returns <code>NULL</code> if no new value exists.</p> <p>If <code>value_type</code> is <code>old</code> or if the <code>command_type</code> of the row LCR is not <code>UPDATE</code>, then the value of the <code>use_old</code> parameter is ignored.</p> <p><code>NULL</code> is not a valid specification for the <code>use_old</code> parameter.</p>

GET_VALUES Member Function

Returns a list of old or new values, depending on the value type specified.

If the command type of the row LCR is `UPDATE`, then specifying `'Y'` for the `use_old` parameter is a convenient way to get the values of all columns.

Syntax

```
MEMBER FUNCTION GET_VALUES(
    value_type  IN VARCHAR2,
    use_old     IN VARCHAR2  DEFAULT 'Y')
RETURN SYS.LCR$_ROW_LIST;
```


Parameters

Table 16–3 *GET_VALUES Procedure Parameters*

Parameter	Description
<code>value_type</code>	The type of values to return. Specify <code>old</code> to return a list of old values. Specify <code>new</code> to return a list of new values.
<code>use_old</code>	<p>If <code>Y</code> and <code>value_type</code> is <code>new</code>, then returns a list of all new values in the LCR. If a new value does not exist in the list, then returns the corresponding old value. Therefore, the returned list contains all existing new values and old values for the new values that do not exist.</p> <p>If <code>N</code> and <code>value_type</code> is <code>new</code>, then returns a list of all new values in the LCR without returning any old values.</p> <p>If <code>value_type</code> is <code>old</code> or if the <code>command_type</code> of the row LCR is not <code>UPDATE</code>, then the value of the <code>use_old</code> parameter is ignored.</p> <p><code>NULL</code> is not a valid specification for the <code>use_old</code> parameter.</p>

Constructing and Processing LCRs Containing LOB Columns

The following are general considerations for row changes involving LOBs in a Streams environment:

- A row change involving a LOB may be captured, propagated, and applied as several LCRs.
- Rules used to evaluate these LCRs must be deterministic, so that either all of the LCRs corresponding to the row change cause a rule in a rule set to evaluate to true, or none of them do.
- Transformations on these LCRs must be deterministic, so that all LCRs corresponding to the row change are transformed in the same way.
- If a user transformation reduces the size of LOB chunks in the LCRs corresponding to a LOB insert without changing the offsets in the LCRs, the resulting LOB will have zero-byte fillers or spaces as documented in the `DBMS_LOB.WRITE` procedure.

The following sections contain information about the requirements you must meet when constructing or processing LOBs and about apply process behavior for LCRs containing LOBs. This section also includes an example that constructs and enqueues LCRs containing LOBs.

Requirements for Constructing and Processing LCRs Containing LOBs

If your environment uses LCRs that contain LOB columns, then you must meet the following requirements when you construct these LCRs or process them with an apply handler or a rule-based transformation:

- The data portion of the LCR LOB column must be of type `VARCHAR2` or `RAW`. A `VARCHAR2` is interpreted as a CLOB, and a `RAW` is interpreted as a BLOB.
- `LOB WRITE`, `LOB ERASE`, and `LOB TRIM` are the only valid command types for out-of-line LOBs.
- For `LOB WRITE`, `LOB ERASE`, and `LOB TRIM` LCRs, the `old_values` collection should be empty or `NULL` and `new_values` should not be empty.
- The `lob_offset` should be a valid value for `LOB WRITE` and `LOB ERASE` LCRs. For all other command types, `lob_offset` should be `NULL`, under the assumption that LOB chunks for that column will follow.
- The `lob_operation_size` should be a valid value for `LOB ERASE` and `LOB TRIM` LCRs. For all other command types, `lob_operation_size` should be `NULL`.
- `LOB TRIM` and `LOB ERASE` are valid command types only for an LCR containing a LOB column with `lob_information` set to `LAST_LOB_CHUNK`.
- `LOB WRITE` is a valid command type only for an LCR containing a LOB column with `lob_information` set to `LAST_LOB_CHUNK` or `LOB_CHUNK`.
- For LOBs with `lob_information` set to `NULL_LOB`, the data portion of the column should be a `NULL` of `VARCHAR2` type (for a CLOB) or a `NULL` of `RAW` type (for a BLOB). Otherwise, it is interpreted as a non-`NULL` inline LOB column.
- Only one LOB column reference with one new chunk is allowed for each `LOB WRITE`, `LOB ERASE`, and `LOB TRIM` LCR.
- The new LOB chunk for a `LOB ERASE` and a `LOB TRIM` LCR should be a `NULL` value encapsulated in a `SYS.AnyData`.

All validation of these requirements is done by an apply process. If these requirements are not met, then an LCR containing a LOB column cannot be applied by an apply process nor processed by an apply handler. In this case, the LCR is moved to an exception queue with the rest of the LCRs in the same transaction.

See Also:

- ["Constructing and Enqueuing LCRs"](#) on page 16-2
- ["Event Processing with an Apply Process"](#) on page 4-3 for more information about apply handlers
- *Oracle9i Application Developer's Guide - Large Objects (LOBs)* for more information about LOBs

Apply Process Behavior for LCRs Containing LOBs

An apply process behaves in the following way when it encounters an LCR that contains a LOB:

- If an LCR whose command type is `INSERT` or `UPDATE` has a new LOB that contains data and the `lob_information` is not `DBMS_LCR.LOB_CHUNK` or `DBMS_LCR.LAST_LOB_CHUNK`, then the data is applied.
- If an LCR whose command type is `INSERT` or `UPDATE` has a new LOB that contains no data, and the `lob_information` is `DBMS_LCR.EMPTY_LOB`, then it is applied as an empty LOB.
- If an LCR whose command type is `INSERT` or `UPDATE` has a new LOB that contains no data, and the `lob_information` is `DBMS_LCR.NULL_LOB` or `DBMS_LCR.INLINE_LOB`, then it is applied as a `NULL`.
- If an LCR whose command type is `INSERT` or `UPDATE` has a new LOB and the `lob_information` is `DBMS_LCR.LOB_CHUNK` or `DBMS_LCR.LAST_LOB_CHUNK`, then any LOB value is ignored. If the command type is `INSERT`, then an empty LOB is inserted into the column under the assumption that LOB chunks will follow. If the command type is `UPDATE`, then the column value is ignored under the assumption that LOB chunks will follow.
- If all of the new columns in an LCR whose command type is `UPDATE` are LOBs whose `lob_information` is `DBMS_LCR.LOB_CHUNK` or `DBMS_LCR.LAST_LOB_CHUNK`, then the update is skipped under the assumption that LOB chunks will follow.
- For any LCR whose command type is `UPDATE` or `DELETE`, old LOB values are ignored.

Example Script for Constructing and Enqueuing LCRs Containing LOBs

The following example illustrates creating a PL/SQL procedure for constructing and enqueuing LCRs containing LOBs. This example assumes that you have prepared your database for Streams by completing the necessary actions in [Chapter 11, "Configuring a Streams Environment"](#). Make sure the Streams administrator who runs this script has EXECUTE privilege on the DBMS_AQ and DBMS_APPLY_ADM packages.

1. [Show Output and Spool Results](#)
2. [Connect as the Streams Administrator](#)
3. [Create a Streams Queue](#)
4. [Create and Start an Apply Process](#)
5. [Create a Schema with Tables Containing LOB Columns](#)
6. [Grant the Streams Administrator Necessary Privileges on the Tables](#)
7. [Create a PL/SQL Procedure to Enqueue LCRs Containing LOBs](#)
8. [Create the do_enq_clob Function to Enqueue CLOBs](#)
9. [Enqueue CLOBs Using the do_enq_clob Function](#)
10. [Check the Spool Results](#)

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the next "END OF SCRIPT" line on page 16-25 into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to all of the databases in the environment.

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

Step 1 Show Output and Spool Results

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

```
*/  
  
SET ECHO ON  
SPOOL lob_construct.out  
  
/*
```

Step 2 Connect as the Streams Administrator

```
*/  
  
SET ECHO ON  
SET FEEDBACK 1  
SET NUMWIDTH 10  
SET LINESIZE 80  
SET TRIMSPOOL ON  
SET TAB OFF  
SET PAGESIZE 100  
SET SERVEROUTPUT ON SIZE 100000  
  
CONNECT strmadmin/strmadminpw  
  
/*
```

Step 3 Create a Streams Queue

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

Step 4 Create and Start an Apply Process

```
*/

BEGIN
  DBMS_APPLY_ADM.CREATE_APPLY(
    queue_name      => 'strmadmin.lobex_queue',
    apply_name      => 'apply_lob',
    apply_captured  => false);
END;
/

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

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

/*
```

Step 5 Create a Schema with Tables Containing LOB Columns

```
*/

CONNECT sys/change_on_install AS SYSDBA

CREATE USER lob_user IDENTIFIED BY Lob_user_pw;
GRANT CONNECT,RESOURCE TO lob_user;

CONNECT lob_user/lob_user_pw

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

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

```
/*
```

Step 6 Grant the Streams Administrator Necessary Privileges on the Tables

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

```
*/
```

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

```
/*
```

Step 7 Create a PL/SQL Procedure to Enqueue LCRs Containing LOBs

```
*/
```

```
CONNECT strmadmin/strmadminpw

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

    eopt          DBMS_AQ.ENQUEUE_OPTIONS_T;
    mprop         DBMS_AQ.MESSAGE_PROPERTIES_T;
    enq_msgid     RAW(16);
    xr_lcr        SYS.LCR$_ROW_RECORD;
BEGIN
    mprop.SENDER_ID := SYS.AQ$_AGENT('strmadmin', NULL, NULL);
    xr_lcr := SYS.LCR$_ROW_RECORD.CONSTRUCT(
        source_database_name => source_dbname,
        command_type         => cmd_type,
        object_owner         => obj_owner,
        object_name          => obj_name,
        old_values            => old_vals,
        new_values            => new_vals);
```

```

-- Enqueue a row lcr
DBMS_AQ.ENQUEUE(
    queue_name      => 'lobex_queue',
    enqueue_options => eopt,
    message_properties => mprop,
    payload         => SYS.AnyData.ConvertObject(xr_lcr),
    msgid          => enq_msgid);
END enq_row_lcr;
/
SHOW ERRORS

/*

```

Step 8 Create the do_enq_clob Function to Enqueue CLOBs

```

*/

-- Description of each variable:
-- src_dbname  : Source database name
-- tab_owner   : Table owner
-- tab_name    : Table name
-- col_name    : Name of the CLOB column
-- new_vals    : SYS.LCR$_ROW_LIST containing primary key and supplementally
--              logged columns
-- clob_data   : CLOB that contains data to be sent
-- offset      : Offset from which data should be sent, default is 1
-- lsize       : Size of data to be sent, default is 0
-- chunk_size  : Size used for creating LOB chunks, default is 2048

CREATE OR REPLACE FUNCTION do_enq_clob(src_dbname    VARCHAR2,
                                       tab_owner     VARCHAR2,
                                       tab_name       VARCHAR2,
                                       col_name       VARCHAR2,
                                       new_vals       SYS.LCR$_ROW_LIST,
                                       clob_data      CLOB,
                                       offset         NUMBER default 1,
                                       lsize          NUMBER default 0,
                                       chunk_size     NUMBER default 2048)

RETURN NUMBER IS

```



```
lob_offset NUMBER; -- maintain lob offset
newunit     SYS.LCR$_ROW_UNIT;
tnewvals    SYS.LCR$_ROW_LIST;
lob_flag    NUMBER;
lob_data    VARCHAR2(32767);
lob_size    NUMBER;
unit_pos    NUMBER;
final_size  NUMBER;
exit_flg    BOOLEAN;
c_size      NUMBER;
i           NUMBER;
BEGIN
lob_size := DBMS_LOB.GETLENGTH(clob_data);
unit_pos := new_vals.count + 1;
tnewvals := new_vals;
c_size   := chunk_size;
i := 0;
-- validate parameters
IF (unit_pos <= 1) THEN
    DBMS_OUTPUT.PUT_LINE('Invalid new_vals list');
    RETURN 1;
END IF;

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

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

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

```

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

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

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

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

exit_flg := FALSE;

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

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

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

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

```

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

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

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

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

IF ((final_size - lob_offset) < c_size) THEN
    c_size := final_size - lob_offset + 1;
END IF;

-- The following output line is for debugging purposes only.
DBMS_OUTPUT.PUT_LINE('Next chunk size : ' || TO_CHAR(c_size));

IF (c_size < 1) THEN
    exit_flg := TRUE;
END IF;

EXIT WHEN exit_flg;

END LOOP;

RETURN 0;
END do_enq_clob;
/

SHOW ERRORS

/*
```

Step 9 Enqueue CLOBs Using the do_enq_clob Function

The DBMS_OUTPUT lines in the following example can be used for debugging purposes if necessary. If they are not needed, then they can be commented out or deleted.

```

*/

SET SERVEROUTPUT ON SIZE 100000
DECLARE
  c1_data CLOB;
  c2_data CLOB;
  c3_data CLOB;
  newunit1 SYS.LCR$_ROW_UNIT;
  newunit2 SYS.LCR$_ROW_UNIT;
  newunit3 SYS.LCR$_ROW_UNIT;
  newunit4 SYS.LCR$_ROW_UNIT;
  newvals SYS.LCR$_ROW_LIST;
  big_data VARCHAR(22000);
  n NUMBER;
BEGIN
  -- Create primary key for LCR$_ROW_UNIT
  newunit1 := SYS.LCR$_ROW_UNIT('A',
                                Sys.AnyData.ConvertNumber(3),
                                NULL,
                                NULL,
                                NULL);

  -- Create empty CLOBs
  newunit2 := sys.lcr$_row_unit('C1',
                                Sys.AnyData.ConvertVarChar2(NULL),
                                DBMS_LCR.EMPTY_LOB,
                                NULL,
                                NULL);
  newunit3 := SYS.LCR$_ROW_UNIT('C2',
                                Sys.AnyData.ConvertVarChar2(NULL),
                                DBMS_LCR.EMPTY_LOB,
                                NULL,
                                NULL);
  newunit4 := SYS.LCR$_ROW_UNIT('C3',
                                Sys.AnyData.ConvertVarChar2(NULL),
                                DBMS_LCR.EMPTY_LOB,
                                NULL,
                                NULL);
  newvals := SYS.LCR$_ROW_LIST(newunit1,newunit2,newunit3,newunit4);

```

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

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

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

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

```

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

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

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

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

```

```
COMMIT;
```

```
END;
```

```
/
```

```
/*
```

Step 10 Check the Spool Results

Check the `lob_construct.out` spool file to ensure that all actions completed successfully after this script completes.

```
*/
```

```
SET ECHO OFF
```

```
SPOOL OFF
```

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

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

```
CONNECT lob_user/lob_user_pw
```

```
EXECUTE DBMS_LOCK.SLEEP(10);
```

```
SELECT a, c1, c2, c3 FROM with_clob ORDER BY a;
```

```
SELECT a, LENGTH(c1), LENGTH(c2), LENGTH(c3) FROM with_clob ORDER BY a;
```

Managing Streams Tags

You can set or get the value of the tags generated by the current session or by an apply process. The following sections describe how to set and get tag values.

- [Managing Streams Tags for the Current Session](#)
- [Managing Streams Tags for an Apply Process](#)

See Also:

- [Chapter 8, "Streams Tags"](#)
- ["Monitoring Streams Tags" on page 17-50](#)

Managing Streams Tags for the Current Session

This section contains instructions for setting and getting the tag for the current session.

Setting the Tag Values Generated by the Current Session

You can set the tag for all redo entries generated by the current session using the `SET_TAG` procedure in the `DBMS_STREAMS` package. For example, to set the tag to the hexadecimal value of '1D' in the current session, run the following procedure:

```
BEGIN
  DBMS_STREAMS.SET_TAG(
    tag => HEXTORAW('1D'));
END;
/
```

After running this procedure, each redo entry generated by DML or DDL statements in the current session will have a tag value of 1D. Running this procedure affects only the current session.

Getting the Tag Value for the Current Session

You can get the tag for all redo entries generated by the current session using the `GET_TAG` procedure in the `DBMS_STREAMS` package. For example, to get the hexadecimal value of the tags generated in the redo entries for the current session, run the following procedure:

```
SET SERVEROUTPUT ON
DECLARE
    raw_tag RAW(2048);
BEGIN
    raw_tag := DBMS_STREAMS.GET_TAG();
    DBMS_OUTPUT.PUT_LINE('Tag Value = ' || RAWTOHEX(raw_tag));
END;
/
```

You can also display the tag value for the current session by querying the `DUAL` view:

```
SELECT DBMS_STREAMS.GET_TAG FROM DUAL;
```

Managing Streams Tags for an Apply Process

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

See Also:

- ["Tags and an Apply Process"](#) on page 8-6 for conceptual information about how tags are used by an apply process and apply handlers
- [Chapter 4, "Streams Apply Process"](#)
- [Chapter 14, "Managing an Apply Process"](#)

Setting the Tag Values Generated by an Apply Process

An apply process generates redo entries when it applies changes to a database or invokes handlers. You can set the default tag for all redo entries generated by an apply process when you create the apply process using the `CREATE_APPLY` procedure in the `DBMS_APPLY_ADM` package, or when you alter an existing apply process using the `ALTER_APPLY` procedure in the `DBMS_APPLY_ADM` package. In both of these procedures, set the `apply_tag` parameter to the value you want to specify for the tags generated by the apply process.

For example, to set the value of the tags generated in the redo log by an existing apply process named `strm01_apply` to the hexadecimal value of '7', run the following procedure:

```
BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name => 'strm01_apply',
    apply_tag  => HEXTORAW('7'));
END;
/
```

After running this procedure, each redo entry generated by the apply process will have a tag value of 7.

Removing the Apply Tag for an Apply Process

You remove the apply tag for an apply process by setting the `remove_apply_tag` parameter to `true` in the `ALTER_APPLY` procedure in the `DBMS_APPLY_ADM` package. Removing the apply tag means that each redo entry generated by the apply process has a `NULL` tag. For example, the following procedure removes the apply tag from an apply process named `strm02_apply`.

```
BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name      => 'strm02_apply',
    remove_apply_tag => true);
END;
/
```

Performing Database Point-in-Time Recovery on a Destination Database

Point-in-time recovery is the recovery of a database to a specified noncurrent time, SCN, or log sequence number. If point-in-time recovery is required at a destination database in a Streams environment, then you must reapply the captured changes that had already been applied after the point-in-time of the recovery.

For each relevant capture process, you can choose either of the following methods to perform point-in-time recovery at a destination database in a Streams environment

- Reset the start SCN for the existing capture process that captures the changes that are applied at the destination database.
- Create a new capture process to capture the changes that must be reapplied at the destination database.

Resetting the start SCN for the capture process is simpler than creating a new capture process. However, if the capture process captures changes that are applied at multiple destination databases, then the changes are resent to all the destination databases, including the ones that did not perform point-in-time recovery. If a change is already applied at a destination database, then it is discarded by the apply process, but you may not want to use the network and computer resources required to resend the changes to multiple destination databases. In this case, you can create and temporarily use a new capture process and a new propagation that propagates changes only to the destination database that was recovered.

The following sections provide instructions for each task:

- [Resetting the Start SCN for the Existing Capture Process to Perform Recovery](#)
- [Creating a New Capture Process to Perform Recovery](#)

If there are multiple apply processes at the destination database where you performed point-in-time recovery, then complete one of the tasks in this section for each apply process.

Neither of these methods should be used if any of the following conditions are true regarding the destination database you are recovering:

- A propagation propagates user-enqueued messages to the destination database. Both of these methods reapply only captured events at the destination database, not user-enqueued events.
- The destination database is also a source database.

- In a directed networks configuration, the destination database is used to propagate events from a capture process to other databases, but the destination database does not apply events from this capture process.

If any of these conditions are true in your environment, then you cannot use the methods described in this section. Instead, you must manually resynchronize the data at all destination databases.

See Also:

- *Oracle9i Backup and Recovery Concepts* for more information about point-in-time recovery
- ["The Start SCN, Captured SCN, and Applied SCN for a Capture Process"](#) on page 2-15
- ["Resetting the Log Sequence Number Where Changes Are Captured"](#) on page 12-13
- ["Directed Networks"](#) on page 3-7

Resetting the Start SCN for the Existing Capture Process to Perform Recovery

If you decide to reset the start SCN for the existing capture process to perform point-in-time recovery, then complete the following steps:

1. If you are not using directed networks between the source database and destination database, then drop the propagation that propagates changes from the source queue at the source database to the destination queue at the destination database. Use the `DROP_PROPAGATION` procedure in the `DBMS_PROPAGATION_ADM` package to drop the propagation.

If you are using directed networks, and there are intermediate databases between the source database and destination database, then drop the propagation at each intermediate database in the path to the destination database, including the propagation at the source database.

Note: You must drop the appropriate propagation(s). Disabling them is not sufficient. You will re-create the propagation(s) in Step 6, and dropping them now ensures that only events created after resetting the start SCN for the capture process are propagated.

See Also: ["Directed Networks"](#) on page 3-7

2. Perform the point-in-time recovery at the destination database.
3. Query for the oldest message number from the source database for the apply process at the destination database. Then, make a note of the results of the query. The oldest message number is the earliest system change number (SCN) that may need to be applied.

The following statement is an example of the query to perform:

```
SELECT APPLY_NAME, OLDEST_MESSAGE_NUMBER FROM DBA_APPLY_PROGRESS;
```

4. Stop the existing capture process using the `STOP_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package.
5. Reset the start SCN of the existing capture process.

To reset the start SCN for an existing capture process, run the `ALTER_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package and set the `start_scn` parameter to the value you recorded from the query in Step 3. For example, to reset the start SCN for a capture process named `strm01_capture` to the value 829381993, run the following `ALTER_CAPTURE` procedure:

```
BEGIN
  DBMS_CAPTURE_ADM.ALTER_CAPTURE(
    capture_name => 'strm01_capture',
    start_scn    => 829381993);
END;
/
```

6. If you are not using directed networks between the source database and destination database, then create a new propagation to propagate changes from the source queue to the destination queue using the `CREATE_PROPAGATION` procedure in the `DBMS_PROPAGATION_ADM` package. Specify the rule set used by the original propagation for the `rule_set_name` parameter when you create the propagation.

If you are using directed networks, and there are intermediate databases between the source database and destination database, then create a new propagation at each intermediate database in the path to the destination database, including the propagation at the source database.

7. Start the existing capture process using the `START_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package.

Creating a New Capture Process to Perform Recovery

If you decide to create a new capture process to perform point-in-time recovery, then complete the following steps:

1. If you are not using directed networks between the source database and destination database, then drop the propagation that propagates changes from the source queue at the source database to the destination queue at the destination database. Use the `DROP_PROPAGATION` procedure in the `DBMS_PROPAGATION_ADM` package to drop the propagation.

If you are using directed networks, and there are intermediate databases between the source database and destination database, then drop the propagation that propagates events between the last intermediate database and the destination database. You do not need to drop the propagations at the other intermediate databases nor at the source database.

Note: You must drop the appropriate propagation. Disabling it is not sufficient.

See Also: ["Directed Networks"](#) on page 3-7

2. Perform the point-in-time recovery at the destination database.
3. Query for the oldest message number from the source database for the apply process at the destination database. Then, make a note of the results of the query. The oldest message number is the earliest system change number (SCN) that may need to be applied.

The following statement is an example of the query to perform:

```
SELECT APPLY_NAME, OLDEST_MESSAGE_NUMBER FROM DBA_APPLY_PROGRESS;
```

4. Create a queue at the source database to be used by the capture process using the `SET_UP_QUEUE` procedure in the `DBMS_STREAMS_ADM` package.

If you are using directed networks, and there are intermediate databases between the source database and destination database, then create a queue at each intermediate database in the path to the destination database, including the new queue at the source database. Do not create a new queue at the destination database.

5. If you are not using directed networks between the source database and destination database, then create a new propagation to propagate changes from the source queue created in Step 4 to the destination queue using the `CREATE_PROPAGATION` procedure in the `DBMS_PROPAGATION_ADM` package. Specify the rule set used by the original propagation for the `rule_set_name` parameter when you create the propagation.

If you are using directed networks, and there are intermediate databases between the source database and destination database, then create a propagation at each intermediate database in the path to the destination database, including the propagation from the source database to the first intermediate database. These propagations propagate changes captured by the capture process you will create in Step 6 between the queues created in Step 4.

6. Create a new capture process at the source database using the `CREATE_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package. Set the `source_queue` parameter to the local queue you created in Step 4, the `rule_set_name` parameter to the rule set used by the original capture process, and the `start_scn` parameter to the value you recorded from the query in Step 3. If the rule set used by the original capture process captures events that should not be sent to the destination database that was recovered, then you can create and use a smaller, customized rule set that shares some rules with the original rule set.
7. Start the capture process you created in Step 6 using the `START_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package.
8. When the oldest message number of the apply process at the recovered database is approaching the capture number of the original capture process at the source database, stop the original capture process using the `STOP_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package.

At the destination database, you can use the following query to determine the oldest message number from the source database for the apply process:

```
SELECT APPLY_NAME, OLDEST_MESSAGE_NUMBER FROM DBA_APPLY_PROGRESS;
```

At the source database, you can use the following query to determine the capture number of the original capture process:

```
SELECT CAPTURE_NAME, CAPTURE_MESSAGE_NUMBER FROM V$STREAMS_CAPTURE;
```

9. When the oldest message number of the apply process at the recovered database is beyond the capture number of the original capture process at the source database, drop the new capture process created in Step 6.

10. If you are not using directed networks between the source database and destination database, then drop the new propagation created in Step 5.

If you are using directed networks, and there are intermediate databases between the source database and destination database, then drop the new propagation at each intermediate database in the path to the destination database, including the new propagation at the source database.

11. If you are not using directed networks between the source database and destination database, then remove the queue created in Step 4.

If you are using directed networks, and there are intermediate databases between the source database and destination database, then drop the new queue at each intermediate database in the path to the destination database, including the new queue at the source database. Do not drop the queue at the destination database.

12. If you are not using directed networks between the source database and destination database, then create a propagation that propagates changes from the original source queue at the source database to the destination queue at the destination database. Use the `CREATE_PROPAGATION` procedure in the `DBMS_PROPAGATION_ADM` package to create the propagation. Specify the rule set used by the original propagation for the `rule_set_name` parameter when you create the propagation.

If you are using directed networks, and there are intermediate databases between the source database and destination database, then re-create the propagation from the last intermediate database to the destination database. You dropped this propagation in Step 1.

13. Start the capture process you stopped in Step 8.

All of the steps after Step 7 can be deferred to a later time, or they can be done as soon as the condition described in Step 8 is met.

Performing Full Database Export/Import on a Database Using Streams

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.

Note: If you want to add a database to an existing Streams environment, then do not use the instructions in this section. Instead, see ["Configuring a Capture-Based Streams Environment"](#) on page 11-15.

See Also:

- ["Setting Export and Import Parameters Relevant to Streams"](#) on page 11-8
- *Oracle9i 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.

After completing this step, do not stop any capture process running on the export database. Step 10 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 10 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 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.

6. If the export database is running one or more capture processes, then wait until the applied SCN of each capture process has reached or exceeded the SCN determined in Step 4.

You can view the applied SCN for each capture process by querying the `APPLIED_SCN` column in the `DBA_CAPTURE` data dictionary view.

7. 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.
8. 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 5, then proceed to Step 9.
9. When the export is complete, transfer the export dump file to the computer running the import database.

10. If the export database is running one or more capture processes, then ensure that all DML and DDL changes on the export database were stopped before the SCN determined in Step 4 by completing the following steps:
 - a. Get the current SCN using the `GET_SYSTEM_CHANGE_NUMBER` function in the `DBMS_FLASHBACK` package. This SCN will be called the new SCN.
 - b. Wait until the capture message number of each capture process has reached or exceeded the new SCN determined in Step a. You can view the capture message number for each capture process by querying the `CAPTURE_MESSAGE_NUMBER` column in the `V$STREAMS_CAPTURE` dynamic performance view.
 - c. Verify that the enqueue message number of each capture process is less than or equal to the SCN determined in Step 4. You can view the enqueue message number for each capture process by querying the `ENQUEUE_MESSAGE_NUMBER` column in the `V$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 11.

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 16-35.

Note: For this verification to be valid, each capture process must have been running uninterrupted since Step 4.

11. 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.
12. Let users access the import database, and shut down the export database.
13. Enable any propagation jobs you disabled in Steps 1 and 7.

If you reset the value of a `message_limit` or `time_limit` capture process parameter in Step 4, then reset these parameters to their original settings.

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:

- [Summary of Streams Static Data Dictionary Views](#)
- [Summary of Streams Dynamic Performance Views](#)
- [Monitoring a Streams Capture Process](#)
- [Monitoring a Streams Queue](#)
- [Monitoring Streams Propagations and Propagation Jobs](#)
- [Monitoring a Streams Apply Process](#)
- [Monitoring Rules and Rule-Based Transformations](#)
- [Monitoring Streams Tags](#)

Note: The Streams tool in Oracle Enterprise Manager is also an excellent way to monitor a Streams environment. See the online help for the Streams tool for more information.

See Also: *Oracle9i Database Reference* for information about the data dictionary views described in this chapter

Summary of Streams Static Data Dictionary Views

The following table lists the Streams static data dictionary views.

Table 17–1 Streams Static Data Dictionary Views

ALL_ Views	DBA_ Views	USER_ Views
ALL_APPLY	DBA_APPLY	No USER_ view
ALL_APPLY_CONFLICT_COLUMNS	DBA_APPLY_CONFLICT_COLUMNS	No USER_ view
ALL_APPLY_DML_HANDLERS	DBA_APPLY_DML_HANDLERS	No USER_ view
ALL_APPLY_ERROR	DBA_APPLY_ERROR	No USER_ view
No ALL_ view	DBA_APPLY_INSTANTIATED_OBJECTS	No USER_ view
ALL_APPLY_KEY_COLUMNS	DBA_APPLY_KEY_COLUMNS	No USER_ view
ALL_APPLY_PARAMETERS	DBA_APPLY_PARAMETERS	No USER_ view
ALL_APPLY_PROGRESS	DBA_APPLY_PROGRESS	No USER_ view
ALL_CAPTURE	DBA_CAPTURE	No USER_ view
ALL_CAPTURE_PARAMETERS	DBA_CAPTURE_PARAMETERS	No USER_ view
ALL_CAPTURE_PREPARED_DATABASE	DBA_CAPTURE_PREPARED_DATABASE	No USER_ view
ALL_CAPTURE_PREPARED_SCHEMAS	DBA_CAPTURE_PREPARED_SCHEMAS	No USER_ view
ALL_CAPTURE_PREPARED_TABLES	DBA_CAPTURE_PREPARED_TABLES	No USER_ view
ALL_EVALUATION_CONTEXT_TABLES	DBA_EVALUATION_CONTEXT_TABLES	USER_EVALUATION_CONTEXT_TABLES
ALL_EVALUATION_CONTEXT_VARS	DBA_EVALUATION_CONTEXT_VARS	USER_EVALUATION_CONTEXT_VARS
ALL_EVALUATION_CONTEXTS	DBA_EVALUATION_CONTEXTS	USER_EVALUATION_CONTEXTS
ALL_PROPAGATION	DBA_PROPAGATION	No USER_ view
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
ALL_STREAMS_GLOBAL_RULES	DBA_STREAMS_GLOBAL_RULES	No USER_ view
ALL_STREAMS_SCHEMA_RULES	DBA_STREAMS_SCHEMA_RULES	No USER_ view
ALL_STREAMS_TABLE_RULES	DBA_STREAMS_TABLE_RULES	No USER_ view

Summary of Streams Dynamic Performance Views

The following list includes the Streams dynamic performance views

- V\$STREAMS_APPLY_COORDINATOR
- V\$STREAMS_APPLY_READER
- V\$STREAMS_APPLY_SERVER
- V\$STREAMS_CAPTURE

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 Set, and Status of Each Capture Process](#)
- [Displaying General Information About a Capture Process](#)
- [Listing the Parameter Settings for a Capture Process](#)
- [Determining the Applied SCN for All Capture Processes in a Database](#)
- [Determining Redo Log Scanning Latency for a Capture Process](#)
- [Determining Event Enqueuing Latency for a Capture Process](#)
- [Determining Which Database Objects Are Prepared for Instantiation](#)
- [Displaying Supplemental Log Groups at a Source Database](#)

See Also:

- [Chapter 2, "Streams Capture Process"](#)
- [Chapter 12, "Managing a Capture Process"](#)

Displaying the Queue, Rule Set, 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 rule set used by the capture process
- The status of the capture process, which may be ENABLE, 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 A20
COLUMN RULE_SET_NAME HEADING 'Capture|Process|Rule Set' FORMAT A15
COLUMN STATUS HEADING 'Capture|Process|Status' FORMAT A15

SELECT CAPTURE_NAME, QUEUE_NAME, RULE_SET_NAME, STATUS FROM DBA_CAPTURE;
```

Your output looks similar to the following:

Capture Process Name	Capture Process Queue	Capture Process Rule Set	Capture Process Status
-----	-----	-----	-----
CAPTURE	STREAMS_QUEUE	RULESET\$_6	ENABLED

Displaying General Information About a Capture Process

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

- The process number (*cpnn*)
- The session identifier
- The serial number of the session
- The current state of the capture process, either INITIALIZING, CAPTURING CHANGES, EVALUATING RULE, ENQUEUEING MESSAGE, SHUTTING DOWN, or CREATING LCR
- The total number of redo entries scanned
- The total number LCRs enqueued

For example, to display this information for a capture process named `capture`, run the following query:

```

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 A17
COLUMN TOTAL_MESSAGES_CAPTURED HEADING 'Redo Entries|Scanned' FORMAT 9999999
COLUMN TOTAL_MESSAGES_ENQUEUED HEADING 'Total|LCRs|Enqueued' FORMAT 999999

SELECT 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.CAPTURE_NAME = 'CAPTURE' AND
       c.SID = s.SID AND
       c.SERIAL# = s.SERIAL#;

```

Your output looks similar to the following:

Capture Process Number	Session ID	Serial Number	State	Redo Entries Scanned	Total LCRs Enqueued
CP01	18	150	CAPTURING CHANGES	56900	7

The number of redo entries scanned may be higher than the number of DML and DDL redo entries that evaluate to `TRUE` for a capture process rule set. Only DML and DDL redo entries that evaluate to `TRUE` for a capture process rule set 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

Listing the Parameter Settings for a Capture Process

The query in this section displays the current setting for each capture process parameter for a particular capture process.

For example, to display the settings for the capture process parameters of a capture process named `capture`, run the following query:

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

SELECT PARAMETER,
       VALUE,
       SET_BY_USER
FROM DBA_CAPTURE_PARAMETERS
WHERE CAPTURE_NAME = 'CAPTURE';
```

Your output looks similar to the following:

Parameter	Value	Set by User?
DISABLE_ON_LIMIT	N	NO
MAXIMUM_SCN	INFINITE	NO
MESSAGE_LIMIT	INFINITE	NO
PARALLELISM	3	YES
STARTUP_SECONDS	0	NO
TIME_LIMIT	INFINITE	NO
TRACE_LEVEL	0	NO
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 2-25
- ["Setting a Capture Process Parameter"](#) on page 12-8

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. This SCN is important because all redo logs must be kept available to a capture process until all transactions within the redo logs have been applied at all downstream databases.

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 999999
```

```
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 a Capture Process

You can find the following information about a 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 a capture process named `capture`:

```
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 ((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
WHERE CAPTURE_NAME = 'CAPTURE';
```

Your output looks similar to the following:

Latency	Current
in Seconds	Since Process
Seconds	Last Status Time
	Event Creation Time
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 a Capture Process

You can find the following information about a 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 a capture process named `capture`:

```
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 (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
WHERE CAPTURE_NAME = 'CAPTURE';
```

Your output looks similar to the following:

Latency	in Event Creation	Message
Seconds	Time	Number
0	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."

Determining Which Database Objects Are Prepared for Instantiation

You prepare a database object 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 all of the database objects in a schema for instantiation.
- `PREPARE_GLOBAL_INSTANTIATION` prepares all of the database objects in a database for instantiation.

To determine which database objects have been prepared for instantiation, query the following corresponding data dictionary views:

- DBA_CAPTURE_PREPARED_TABLES
- DBA_CAPTURE_PREPARED_SCHEMAS
- DBA_CAPTURE_PREPARED_DATABASE

For example, to list all of the tables that have been prepared for instantiation, the SCN for the time when each table was prepared, and the time when each table was prepared, run the following query:

```
COLUMN TABLE_OWNER HEADING 'Table Owner' FORMAT A15
COLUMN TABLE_NAME HEADING 'Table Name' FORMAT A15
COLUMN SCN HEADING 'Instantiation SCN' FORMAT 999999
COLUMN TIMESTAMP HEADING 'Time Ready for|Instantiation'

SELECT TABLE_OWNER,
       TABLE_NAME,
       SCN,
       TO_CHAR(TIMESTAMP, 'HH24:MI:SS MM/DD/YY') TIMESTAMP
FROM DBA_CAPTURE_PREPARED_TABLES;
```

Your output looks similar to the following:

Table Owner	Table Name	Instantiation SCN	Time Ready for Instantiation
HR	COUNTRIES	196655	12:59:30 02/28/02
HR	DEPARTMENTS	196658	12:59:30 02/28/02
HR	EMPLOYEES	196659	12:59:30 02/28/02
HR	JOBS	196660	12:59:30 02/28/02
HR	JOB_HISTORY	196661	12:59:30 02/28/02
HR	LOCATIONS	196662	12:59:30 02/28/02
HR	REGIONS	196664	12:59:30 02/28/02

See Also: ["Preparing Database Objects for Instantiation at a Source Database"](#) on page 12-11

Displaying Supplemental Log Groups at a Source Database

Supplemental logging places additional column data into a redo log whenever an UPDATE operation is performed. The capture process captures this additional information and places it in LCRs. An apply process that applies captured LCRs may need this additional information to schedule or apply changes correctly.

To check whether one or more log groups are specified for the table at the source database, run the following query:

```
COLUMN LOG_GROUP_NAME HEADING 'Log Group' FORMAT A20
COLUMN TABLE_NAME HEADING 'Table' FORMAT A20
COLUMN ALWAYS HEADING 'Type of Log Group' FORMAT A30

SELECT
    LOG_GROUP_NAME,
    TABLE_NAME,
    DECODE(ALWAYS,
           'ALWAYS', 'Unconditional',
           NULL,    'Conditional') ALWAYS
FROM DBA_LOG_GROUPS;
```

Your output looks similar to the following:

Log Group	Table	Type of Log Group
LOG_GROUP_DEP_PK	DEPARTMENTS	Unconditional
LOG_GROUP_JOBS_CR	JOBS	Conditional

To list the columns in a particular log group, query the DBA_LOG_GROUP_COLUMNS data dictionary view. You can also query the V\$DATABASE dynamic performance view to display supplemental logging specified at the database level.

See Also:

- ["Supplemental Logging in a Streams Environment"](#) on page 2-11
- ["Specifying Supplemental Logging at a Source Database"](#) on page 12-9

Monitoring a Streams Queue

The following sections contain queries that you can run to display information about a Streams queue:

- [Displaying the Streams Queues in a Database](#)
- [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 13, "Managing Staging and Propagation"](#)

Displaying the Streams Queues in a Database

Streams queues are of object type `SYS.AnyData`. To display all of the Streams queues in a database, run the following query:

```
COLUMN OWNER HEADING 'Owner' FORMAT A10
COLUMN NAME HEADING 'Queue Name' FORMAT A25
COLUMN QUEUE_TABLE HEADING 'Queue Table' FORMAT A20
COLUMN USER_COMMENT HEADING 'Comment' FORMAT A20

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
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 Streams queue.

See Also: ["Managing Streams Queues"](#) on page 13-2

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_queue_table` 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_QUEUE_TABLE;
```

Your output looks similar to the following:

Message ID	Message State	Consumer
99315B276CFA1872E034080020AE3E0A	PROCESSED	APPLY_OE
99315B276CFB1872E034080020AE3E0A	PROCESSED	APPLY_OE
99315B276CFA1872E034080020AE3E0A	READY	EXPLICIT_DQ
99315B276CFB1872E034080020AE3E0A	READY	EXPLICIT_DQ

Note: This query lists only user-enqueued events, not captured events.

See Also: [Chapter 19, "Streams Messaging Example"](#) for an example that enqueues the events shown in this example into a Streams queue

Viewing the Contents of User-Enqueued Events in a Queue

In a Streams queue, to view the contents of a payload that is encapsulated within a `SYS.AnyData` payload, you query the queue table using the `Accessdata_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"](#) on page 13-19 for an example that enqueues the events shown in the queries in this section into a Streams 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;
```

Then, 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 a Propagation](#)
- [Determining the Rule Set for a Propagation](#)
- [Displaying the Schedule for a Propagation Job](#)
- [Determining the Total Number of Events and Bytes Propagated](#)

See Also:

- [Chapter 3, "Streams Staging and Propagation"](#)
- ["Managing Streams Propagations and Propagation Jobs" on page 13-7](#)

Determining the Source Queue and Destination Queue for a Propagation

You can determine the source queue and destination queue for a propagation by querying the `DBA_PROPAGATION` data dictionary view at the database that contains the source queue.

For example, the following query displays the following information for a propagation named `dbst1_to_dbst2`:

- 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
WHERE 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

Determining the Rule Set for a Propagation

The following query displays the owner and name of a rule set used by a propagation named `dbst1_to_dbst2`:

```
COLUMN RULE_SET_OWNER HEADING 'Rule Set Owner' FORMAT A35
COLUMN RULE_SET_NAME HEADING 'Rule Set Name' FORMAT A35

SELECT RULE_SET_OWNER, RULE_SET_NAME
FROM DBA_PROPAGATION
WHERE PROPAGATION_NAME = 'DBS1_TO_DBS2';
```

Your output looks similar to the following:

Rule Set Owner	Rule Set Name
STRMADMIN	RULESET\$_3

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 `db1_to_db2`:

- 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:

Start Date	Duration	Next	Latency	Status	Process	Number of
	in Seconds	Time	in Seconds			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 3-21 for more information about the default propagation schedule for a Streams propagation job
- ["Is the Propagation Job Used by a Propagation Enabled?"](#) on page 18-6 if the propagation job is disabled
- *Oracle9i Application Developer's Guide - Advanced Queuing and Oracle9i Database Reference* for more information about the `DBA_QUEUE_SCHEDULES` data dictionary view

Determining the Total Number of Events and Bytes Propagated

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 a propagation schedule associated with a particular propagation job:

- 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

For example, to display this information for a propagation job used by a propagation named `dbst1_to_dbst2`, run the following query at the database that contains the source queue:

```
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 s.TOTAL_TIME, s.TOTAL_NUMBER, s.TOTAL_BYTES
       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:

```
Total Time Executing
           in Seconds Total Events Propagated Total Bytes Propagated
-----
                           65                               71                               46536
```

See Also: *Oracle9i Application Developer's Guide - Advanced Queuing* and *Oracle9i 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:

- [Displaying General Information About Each Apply Process](#)
- [Listing the Parameter Settings for an Apply Process](#)
- [Displaying Information About Apply Handlers](#)
- [Displaying the Substitute Key Columns Specified at a Destination Database](#)
- [Displaying Information About Update Conflict Handlers for a Destination Database](#)
- [Determining the Tables for Which an Instantiation SCN Has Been Set](#)
- [Displaying Information About the Reader Server for an Apply Process](#)
- [Determining Capture to Dequeue Latency for an Event](#)
- [Displaying Information About the Coordinator Process](#)
- [Determining the Capture to Apply Latency for an Event](#)
- [Displaying Information About the Apply Servers for an Apply Process](#)
- [Displaying Effective Apply Parallelism for an Apply Process](#)
- [Checking for Apply Errors](#)
- [Displaying Detailed Information About Apply Errors](#)

See Also:

- [Chapter 4, "Streams Apply Process"](#)
- [Chapter 14, "Managing an Apply Process"](#)

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 name of the queue used by the apply process
- The name of the rule set used by the apply process
- 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 status of the apply process, which may be 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 'Apply|Process|Rule Set' FORMAT A15
COLUMN APPLY_CAPTURED HEADING 'Type of|Events|Applied' FORMAT A15
COLUMN STATUS HEADING 'Apply|Process|Status' FORMAT A8

SELECT APPLY_NAME,
       QUEUE_NAME,
       RULE_SET_NAME,
       DECODE(APPLY_CAPTURED,
              'YES', 'Captured',
              'NO', 'User-Enqueued') APPLY_CAPTURED,
       STATUS
FROM DBA_APPLY;
```

Your output looks similar to the following:

Apply Process Name	Apply Process Queue	Apply Process Rule Set	Type of Events Applied	Apply Process Status
APPLY_OE	OE_QUEUE	APPLY_OE_RS	User-Enqueued	ENABLED
APPLY	OE_QUEUE	RULESET\$_4	Captured	DISABLED

Listing the Parameter Settings for an Apply Process

The query in this section displays the current setting for each apply process parameter for a particular apply process.

For example, to display the settings for the apply process parameters of an apply process named `strm01_apply`, run the following query:

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

SELECT PARAMETER,
       VALUE,
       SET_BY_USER
FROM DBA_APPLY_PARAMETERS
WHERE APPLY_NAME = 'STRM01_APPLY';
```

Your output looks similar to the following:

Parameter	Value	Set by User?
COMMIT_SERIALIZATION	FULL	NO
DISABLE_ON_ERROR	Y	YES
DISABLE_ON_LIMIT	N	NO
MAXIMUM_SCN	INFINITE	NO
PARALLELISM	1	NO
STARTUP_SECONDS	0	NO
TIME_LIMIT	INFINITE	NO
TRACE_LEVEL	0	NO
TRANSACTION_LIMIT	INFINITE	NO
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 4-34
- ["Setting an Apply Process Parameter"](#) on page 14-11

Displaying Information About Apply Handlers

This section contains queries that display information about apply process DML handlers, DDL handlers, and error handlers.

See Also: ["Event Processing with an Apply Process"](#) on page 4-3

Displaying All of the DML and Error Handlers for Local Apply

When you specify a local DML or error handler using the `SET_DML_HANDLER` procedure in the `DBMS_APPLY_ADM` package at a destination database, the handler is run for all apply processes in the database that apply changes locally, when appropriate. DML and error handlers are run for a specified operation on a specific table.

To display the DML or 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 A9
COLUMN USER_PROCEDURE HEADING 'Handler Procedure' FORMAT A40
COLUMN ERROR_HANDLER HEADING 'Type of|Handler' FORMAT A10

SELECT OBJECT_OWNER,
       OBJECT_NAME,
       OPERATION_NAME,
       USER_PROCEDURE,
       DECODE(ERROR_HANDLER,
              'Y', 'Error',
              'N', 'DML') ERROR_HANDLER
FROM DBA_APPLY_DML_HANDLERS
WHERE APPLY_DATABASE_LINK IS NULL
ORDER BY OBJECT_OWNER, OBJECT_NAME, ERROR_HANDLER;
```

Your output looks similar to the following:

Table Owner	Table Name	Operation	Handler Procedure	Type of Handler
HR	LOCATIONS	UPDATE	STRMADMIN.HISTORY_DML	DML
HR	REGIONS	INSERT	STRMADMIN.ERRORS_PKG.REGIONS_PK_ERROR	Error

Note: You can also specify DML handlers to process changes for remote non-Oracle databases. This query does not display such DML handlers because it lists a DML handler only if the `APPLY_DATABASE_LINK` column is `NULL` for a DML handler.

See Also:

- ["Managing a DML Handler"](#) on page 14-14
- ["Managing an Error Handler"](#) on page 14-21

Displaying the DDL Handler and Message Handler for Each Apply Process

To display the DDL handler and message handler for each apply process in a database, run the following query:

```
COLUMN APPLY_NAME HEADING 'Apply Process Name' FORMAT A20
COLUMN DDL_HANDLER HEADING 'DDL Handler' FORMAT A20
COLUMN MESSAGE_HANDLER HEADING 'Message Handler' FORMAT A20

SELECT APPLY_NAME, DDL_HANDLER, MESSAGE_HANDLER FROM DBA_APPLY;
```

Your output looks similar to the following:

Apply Process Name	DDL Handler	Message Handler
APPLY	oe.ddl_handler	
APPLY_OE		oe.msg_handler

See Also:

- ["Managing the DDL Handler for an Apply Process"](#) on page 14-18
- ["Managing the Message Handler for an Apply Process"](#) on page 14-13

Displaying the Substitute Key Columns Specified at a Destination Database

You can designate a substitute key at a destination database, which is a column or set of columns that Oracle can use to identify rows in the table during apply. Substitute key columns can be used to specify key columns for a table that has no primary key, or they can be used instead of a table's primary key when the table is processed by any apply process at a destination database.

To display all of the substitute key columns specified at a destination database, run the following query:

```
COLUMN OBJECT_OWNER HEADING 'Table Owner' FORMAT A20
COLUMN OBJECT_NAME HEADING 'Table Name' FORMAT A20
COLUMN COLUMN_NAME HEADING 'Substitute Key Name' FORMAT A20
COLUMN APPLY_DATABASE_LINK HEADING 'Database Link|for Remote|Apply' FORMAT A15

SELECT OBJECT_OWNER, OBJECT_NAME, COLUMN_NAME, APPLY_DATABASE_LINK
FROM DBA_APPLY_KEY_COLUMNS
ORDER BY APPLY_DATABASE_LINK, OBJECT_OWNER, OBJECT_NAME;
```

Your output looks similar to the following:

Table Owner	Table Name	Substitute Key Name	Database Link for Remote Apply
HR	DEPARTMENTS	DEPARTMENT_NAME	
HR	DEPARTMENTS	LOCATION_ID	
HR	EMPLOYEES	FIRST_NAME	
HR	EMPLOYEES	LAST_NAME	
HR	EMPLOYEES	HIRE_DATE	

Note: This query shows the database link in the last column if the substitute key columns are for a remote non-Oracle database. The last column is NULL if a substitute key column is specified for the local destination database.

See Also:

- ["Substitute Key Columns"](#) on page 4-11
- ["Managing the Substitute Key Columns for a Table"](#) on page 14-27

Displaying Information About Update Conflict Handlers for a Destination Database

When you specify an update conflict handler using the `SET_UPDATE_CONFLICT_HANDLER` procedure in the `DBMS_APPLY_ADM` package, the update conflict handler is run for all apply processes in the database, when a relevant conflict occurs.

The query in this section displays all of the columns for which conflict resolution has been specified using a prebuilt update conflict handler. That is, it shows the columns in all of the column lists specified in the database. This query also shows the type of prebuilt conflict handler specified and the resolution column specified for the column list.

To display information about all of the update conflict handlers in a database, run the following query:

```
COLUMN OBJECT_OWNER HEADING 'Table|Owner' FORMAT A5
COLUMN OBJECT_NAME HEADING 'Table Name' FORMAT A12
COLUMN METHOD_NAME HEADING 'Method' FORMAT A12
COLUMN RESOLUTION_COLUMN HEADING 'Resolution|Column' FORMAT A13
COLUMN COLUMN_NAME HEADING 'Column Name' FORMAT A30

SELECT OBJECT_OWNER,
       OBJECT_NAME,
       METHOD_NAME,
       RESOLUTION_COLUMN,
       COLUMN_NAME
FROM DBA_APPLY_CONFLICT_COLUMNS
ORDER BY OBJECT_OWNER, OBJECT_NAME, RESOLUTION_COLUMN;
```

Your output looks similar to the following:

Table Owner	Table Name	Method	Resolution Column	Column Name
HR	COUNTRIES	MAXIMUM	TIME	COUNTRY_NAME
HR	COUNTRIES	MAXIMUM	TIME	REGION_ID
HR	COUNTRIES	MAXIMUM	TIME	TIME
HR	DEPARTMENTS	MAXIMUM	TIME	DEPARTMENT_NAME
HR	DEPARTMENTS	MAXIMUM	TIME	LOCATION_ID
HR	DEPARTMENTS	MAXIMUM	TIME	MANAGER_ID
HR	DEPARTMENTS	MAXIMUM	TIME	TIME

See Also:

- [Chapter 7, "Streams Conflict Resolution"](#)
- ["Managing Streams Conflict Resolution" on page 14-29](#)

Determining the Tables for Which an Instantiation SCN Has Been Set

An instantiation SCN is set at a destination database. It controls which captured LCRs for a table are ignored by an apply process and which captured LCRs for a database object are applied by an apply process. If the commit SCN of an LCR for a table from a source database is less than or equal to the instantiation SCN for that table at a destination database, then the apply process at the destination database discards the LCR. Otherwise, the apply process applies the LCR.

The following query lists each table for which an instantiation SCN has been set at a destination database and the instantiation SCN for each table:

```
COLUMN SOURCE_DATABASE HEADING 'Source Database' FORMAT A15
COLUMN SOURCE_OBJECT_OWNER HEADING 'Object Owner' FORMAT A15
COLUMN SOURCE_OBJECT_NAME HEADING 'Object Name' FORMAT A15
COLUMN INSTANTIATION_SCN HEADING 'Instantiation SCN' FORMAT 999999

SELECT SOURCE_DATABASE,
       SOURCE_OBJECT_OWNER,
       SOURCE_OBJECT_NAME,
       INSTANTIATION_SCN
FROM DBA_APPLY_INSTANTIATED_OBJECTS;
```

Your output looks similar to the following:

Source Database	Object Owner	Object Name	Instantiation SCN
DBS1.NET	HR	REGIONS	196660
DBS1.NET	HR	COUNTRIES	196660
DBS1.NET	HR	LOCATIONS	196660

See Also: ["Setting Instantiation SCNs at a Destination Database" on page 14-35](#)

Displaying Information About the Reader Server for an 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 a particular 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.

For example, to display this information for an apply process named `apply`, run the following query:

```
COLUMN APPLY_CAPTURED HEADING 'Apply Type' FORMAT A22
COLUMN PROCESS_NAME HEADING 'Process Name' FORMAT A12
COLUMN STATE HEADING 'State' FORMAT A17
COLUMN TOTAL_MESSAGES_DEQUEUED HEADING 'Total Events Dequeued' FORMAT 99999999

SELECT 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.APPLY_NAME = 'APPLY' AND
      r.SID = s.SID AND
      r.SERIAL# = s.SERIAL# AND
      r.APPLY_NAME = ap.APPLY_NAME;
```

Your output looks similar to the following:

Apply Type	Process Name	State	Total Events Dequeued
Captured LCRS	P000	DEQUEUE MESSAGES	3803

Determining Capture to Dequeue Latency for an Event

The query in this section displays the following information about the last event dequeued by a particular 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 was 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.

For example, to display the capture and propagation latency for the last captured event dequeued by an apply process named `apply`, run the following query:

```
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 (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
WHERE APPLY_NAME = 'APPLY';
```

Your output looks similar to the following:

```
Latency
   in
Seconds Event Creation    Last Dequeue Time    Dequeued
-----
36 10:56:51 03/01/02 10:57:27 03/01/02    253962
```

Displaying Information About the Coordinator Process

A coordinator process gets transactions from the reader server and passes these transactions to apply servers. The coordinator process name is `apnn`, where `nn` is a coordinator process number.

The query in this section displays the following information about the coordinator process for a particular apply process:

- The number of the coordinator in the process name (`apnn`)
- 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 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 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 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
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
```

```

SELECT SUBSTR(s.PROGRAM, INSTR(S.PROGRAM, '(' )+1, 4) PROCESS_NAME,
       c.SID,
       c.SERIAL#,
       c.STATE,
       c.TOTAL_RECEIVED,
       c.TOTAL_APPLIED,
       c.TOTAL_ERRORS
FROM V$STREAMS_APPLY_COORDINATOR c, V$SESSION s
WHERE c.APPLY_NAME = 'APPLY' AND
       c.SID = s.SID AND
       c.SERIAL# = s.SERIAL#;

```

Your output looks similar to the following:

Coordinator Process Name	Session Session ID	Serial Serial Number	State	Total Trans Received	Total Trans Applied	Total Apply Errors
AP01	11	40	APPLYING	78	73	2

Determining the Capture to Apply Latency for an Event

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 a particular apply process:

- 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

To display the capture to apply latency using the V\$STREAMS_APPLY_COORDINATOR view for an event applied by an apply process named `apply`, run the following query:

```

COLUMN 'Latency in Seconds' FORMAT 999999
COLUMN 'Event Creation' FORMAT A17
COLUMN 'Apply Time' FORMAT A17
COLUMN 'Applied Message Number' FORMAT 999999

SELECT (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 "Applied Message Number"
FROM V$STREAMS_APPLY_COORDINATOR
WHERE APPLY_NAME = 'APPLY';
    
```

Your output looks similar to the following:

Latency in Seconds	Event Creation	Apply Time	Applied Message Number
36	10:56:51 03/01/02	10:57:27 03/01/02	253962

Example DBA_APPLY_PROGRESS Query for Latency

To display the capture to apply latency using the DBA_APPLY_PROGRESS view for an event applied by an apply process named `apply`, run the following query:

```
COLUMN 'Latency in Seconds' FORMAT 999999
COLUMN 'Event Creation' FORMAT A17
COLUMN 'Apply Time' FORMAT A17
COLUMN 'Applied Message Number' FORMAT 999999

SELECT (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 "Applied Message Number"
FROM DBA_APPLY_PROGRESS
WHERE APPLY_NAME = 'APPLY';
```

Your output looks similar to the following:

Latency in Seconds	Event Creation	Apply Time	Applied Message Number
38	10:50:09 03/01/02	10:50:47 03/01/02	253678

Displaying Information About the Apply Servers for an 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 a particular apply process:

- The type of events applied by each apply server, either captured LCRs or user-enqueued messages
- 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`, or `WAIT FOR NEXT CHUNK`. See `V$STREAMS_APPLY_SERVER` in the *Oracle9i 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.

For example, to display this information for an apply process named `apply`, run the following query:

```

COLUMN APPLY_CAPTURED HEADING 'Apply Type' 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 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_ASSIGNED,
       r.TOTAL_MESSAGES_APPLIED
FROM V$STREAMS_APPLY_SERVER R, V$SESSION S, DBA_APPLY AP
WHERE r.APPLY_NAME = 'APPLY' AND
      r.SID = s.SID AND
      r.SERIAL# = s.SERIAL# AND
      r.APPLY_NAME = ap.APPLY_NAME
ORDER BY r.SERVER_ID;

```

Your output looks similar to the following:

Apply Type	Process Name	State	Total Transactions Assigned	Total Events Applied
Captured LCRs	P001	IDLE	94	2141
Captured LCRs	P002	IDLE	12	276
Captured LCRs	P003	IDLE	0	0

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 restarted. If the `parallelism` setting for an apply process is higher than the

effective parallelism for the apply process, then consider lowering the parallelism setting.

Checking for Apply Errors

To check for apply errors, run the following query:

```
COLUMN APPLY_NAME HEADING 'Apply|Process|Name' FORMAT A8
COLUMN SOURCE_DATABASE HEADING 'Source|Database' FORMAT A8
COLUMN LOCAL_TRANSACTION_ID HEADING 'Local|Transaction|ID' FORMAT A11
COLUMN ERROR_MESSAGE HEADING 'Error Message' FORMAT A50

SELECT APPLY_NAME, SOURCE_DATABASE, LOCAL_TRANSACTION_ID, ERROR_MESSAGE
FROM DBA_APPLY_ERROR;
```

If there are any apply errors, then your output looks similar to the following:

Apply Process Name	Source Database	Local Transaction ID	Error Message
APPLY	DBS1.NET	5.4.312	ORA-00001: unique constraint (HR.JOB_ID_PK) violated

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:

- ["Exception Queues"](#) on page 4-37
- ["Managing Apply Errors"](#) on page 14-33
- ["Considerations for Applying DML Changes to Tables"](#) on page 4-10 for information about the possible causes of apply errors
- ["Managing Streams Tags for the Current Session"](#) on page 16-26

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 exception queues 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 All Exception Queues](#)
5. [Create a Procedure that Prints All the Error LCRs for a Transaction](#)

Note: These scripts display only the first 255 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.

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. 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 value types.

```
CREATE OR REPLACE PROCEDURE print_any(data IN SYS.AnyData) IS
  tn VARCHAR2(61);
  str VARCHAR2(255);
  chr CHAR(255);
  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(str);
  ELSIF tn = 'SYS.CHAR' then
    res := data.GETCHAR(chr);
    DBMS_OUTPUT.PUT_LINE(chr);
  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(RAWTOHEX(rw));
  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 17-38.

```

CREATE OR REPLACE PROCEDURE print_lcr(lcr IN SYS.ANYDATA) IS
  typenm  VARCHAR2(61);
  ddlcr   SYS.LCR$_DDL_RECORD;
  proclcr SYS.LCR$_PROCEDURE_RECORD;
  rowlcr  SYS.LCR$_ROW_RECORD;
  res     NUMBER;
  newlist SYS.LCR$_ROW_LIST;
  oldlist SYS.LCR$_ROW_LIST;
  ddl_text CLOB;
BEGIN
  typenm := lcr.GETTYPENAME();
  DBMS_OUTPUT.PUT_LINE('type name: ' || typenm);
  IF (typenm = 'SYS.LCR$_DDL_RECORD') THEN
    res := lcr.GETOBJECT(ddlcr);
    DBMS_OUTPUT.PUT_LINE('source database: ' ||
      ddlcr.GET_SOURCE_DATABASE_NAME);
    DBMS_OUTPUT.PUT_LINE('owner: ' || ddlcr.GET_OBJECT_OWNER);
    DBMS_OUTPUT.PUT_LINE('object: ' || ddlcr.GET_OBJECT_NAME);
    DBMS_OUTPUT.PUT_LINE('is tag null: ' || ddlcr.IS_NULL_TAG);
    DBMS_LOB.CREATETEMPORARY(ddl_text, TRUE);
    ddlcr.GET_DDL_TEXT(ddl_text);
    DBMS_OUTPUT.PUT_LINE('ddl: ' || ddl_text);
    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;
  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;
ELSE
  DBMS_OUTPUT.PUT_LINE('Non-LCR Message with type ' || typem);
END IF;
END print_lcr;
/

```

Step 4 Create a Procedure That Prints All the LCRs in All Exception Queues

The following procedure prints all of the LCRs in all of the exception queues in the database. It calls the `print_lcr` procedure created in ["Create a Procedure That Prints a Specified LCR"](#) on page 17-39.

```

CREATE OR REPLACE PROCEDURE print_errors IS
  CURSOR c IS
    SELECT LOCAL_TRANSACTION_ID,
           SOURCE_DATABASE,
           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);
  msgcnt NUMBER;
  errnum NUMBER := 0;
  errno  NUMBER;
  errmsg VARCHAR2(128);
  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;
    errmsg := r.ERROR_MESSAGE;
    errno  := r.ERROR_NUMBER;
  
```

```

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 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 an exception queue for a particular transaction. It calls the `print_lcr` procedure created in ["Create a Procedure That Prints a Specified LCR"](#) on page 17-39.

```

CREATE OR REPLACE PROCEDURE print_transaction(ltxnid IN VARCHAR2) IS
    i      NUMBER;
    txnid  VARCHAR2(30);
    source VARCHAR2(128);
    msgcnt NUMBER;
    errno  NUMBER;
    errmsg VARCHAR2(128);
    lcr    SYS.ANYDATA;
BEGIN
    SELECT LOCAL_TRANSACTION_ID,
           SOURCE_DATABASE,
           MESSAGE_COUNT,
           ERROR_NUMBER,
           ERROR_MESSAGE
    INTO txnid, source, 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 Number: ' || errno);
DBMS_OUTPUT.PUT_LINE('----Message Text: ' || ermsg);
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 the Streams Rules Used by a Streams Process or Propagation](#)
- [Displaying the Condition for a Streams Rule](#)
- [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 the Rule-Based Transformations in a Rule Set](#)

See Also:

- [Chapter 5, "Rules"](#)
- [Chapter 6, "How Rules Are Used In Streams"](#)
- [Chapter 15, "Managing Rules and Rule-Based Transformations"](#)

Displaying the Streams Rules Used by a Streams Process or Propagation

Streams rules are rules created using the `DBMS_STREAMS_ADM` package for a capture process, propagation, or apply process. These rules determine behavior of the capture process, propagation, or apply process. For example, if a capture rule evaluates to `TRUE` for DML changes to the `hr.employees` table, then the capture process captures DML changes to this table.

You query the following data dictionary views to display Streams rules:

- `ALL_STREAMS_GLOBAL_RULES`
- `DBA_STREAMS_GLOBAL_RULES`
- `ALL_STREAMS_SCHEMA_RULES`
- `DBA_STREAMS_SCHEMA_RULES`
- `ALL_STREAMS_TABLE_RULES`
- `DBA_STREAMS_TABLE_RULES`

Note: These views display only rules created using the `DBMS_STREAMS_ADM` package or the Streams tool in Oracle Enterprise Manager. These views 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.

For example, the following query displays all of the schema rules for an apply process named `strm01_apply`:

```
COLUMN SCHEMA_NAME HEADING 'Schema|Name' FORMAT A10
COLUMN SOURCE_DATABASE HEADING 'Source' FORMAT A10
COLUMN RULE_TYPE HEADING 'Rule Type' FORMAT A10
COLUMN RULE_NAME HEADING 'Rule Name' FORMAT A10
COLUMN RULE_OWNER HEADING 'Rule Owner' FORMAT A10
COLUMN INCLUDE_TAGGED_LCR HEADING 'Apply|Tagged|LCRs?' FORMAT A15
```

```
SELECT SCHEMA_NAME, SOURCE_DATABASE, RULE_TYPE,
       RULE_NAME, RULE_OWNER, INCLUDE_TAGGED_LCR
FROM DBA_STREAMS_SCHEMA_RULES
WHERE STREAMS_NAME = 'STRM01_APPLY' AND STREAMS_TYPE = 'APPLY';
```

Your output looks similar to the following:

Schema Name	Source	Rule Type	Rule Name	Rule Owner	Apply Tagged LCRs?
HR	DBS1.NET	DML	HR1	STRMADMIN	NO

These results show that the apply process applies LCRs containing DML changes to the `hr` schema that originated at the `db1.net` database. The rule in the apply process rule set that instructs the apply process to apply these changes is owned by the `strmadmin` user and is named `hr1`. Also, the apply process applies these changes only if the tag in the LCR is `NULL`.

See Also: ["System-Created Rules"](#) on page 6-3

Displaying the Condition for a Streams Rule

If you know the name and level of a Streams rule, then you can display its rule condition. The level is either global, schema, or table.

For example, consider the rule returned by the query in ["Displaying the Streams Rules Used by a Streams Process or Propagation"](#) on page 17-43. The name of the Streams schema rule is `hr1`, and you can display its condition by running the following query:

```
SELECT RULE_CONDITION "Schema Rule Condition"
FROM DBA_STREAMS_SCHEMA_RULES
WHERE RULE_NAME = 'HR1' AND
       RULE_OWNER = 'STRMADMIN';
```

Your output looks similar to the following:

```
Schema 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 6-3

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 A15
COLUMN RULE_SET_NAME HEADING 'Rule Set Name' FORMAT A15
COLUMN RULE_SET_EVAL_CONTEXT_OWNER HEADING 'Eval Context|Owner' FORMAT A12
COLUMN RULE_SET_EVAL_CONTEXT_NAME HEADING 'Eval Context Name' FORMAT A27

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 Owner	Rule Set Name	Eval Context 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

See Also: ["Rule Evaluation Context"](#) on page 5-5

Displaying Information About the Tables Used by an Evaluation Context

The following query displays information about the tables used by an evaluation context named `evalctx`, which is owned by the `support` user:

```
COLUMN TABLE_ALIAS HEADING 'Table Alias' FORMAT A20
COLUMN TABLE_NAME HEADING 'Table Name' FORMAT A40
```

```
SELECT TABLE_ALIAS,
       TABLE_NAME
FROM   DBA_EVALUATION_CONTEXT_TABLES
WHERE  EVALUATION_CONTEXT_OWNER = 'SUPPORT' AND
       EVALUATION_CONTEXT_NAME = 'EVALCTX';
```

Your output looks similar to the following:

Table Alias	Table Name
PROB	problems

See Also: ["Rule Evaluation Context"](#) on page 5-5

Displaying Information About the Variables Used in an Evaluation Context

The following query displays information about the variables used by an evaluation context named `evalctx`, which is owned by the `support` user:

```
COLUMN VARIABLE_NAME HEADING 'Variable Name' FORMAT A15
COLUMN VARIABLE_TYPE HEADING 'Variable Type' FORMAT A15
COLUMN VARIABLE_VALUE_FUNCTION HEADING 'Variable Value|Function' FORMAT A20
COLUMN VARIABLE_METHOD_FUNCTION HEADING 'Variable Method|Function' FORMAT A20

SELECT VARIABLE_NAME,
       VARIABLE_TYPE,
       VARIABLE_VALUE_FUNCTION,
       VARIABLE_METHOD_FUNCTION
FROM   DBA_EVALUATION_CONTEXT_VARS
WHERE  EVALUATION_CONTEXT_OWNER = 'SUPPORT' AND
       EVALUATION_CONTEXT_NAME = 'EVALCTX';
```

Your output looks similar to the following:

Variable Name	Variable Type	Variable Value Function	Variable Method Function
CURRENT_TIME	DATE	timefunc	

See Also: ["Rule Evaluation Context"](#) on page 5-5

Displaying All of the Rules in a Rule Set

The query in this section displays the following information about all of the 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:

Rule Owner	Rule Name	Eval Context Name	Eval Context 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 6-3](#)

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 the Rule-Based Transformations in a Rule Set

In Streams, a rule-based transformation is specified in a rule action context that has the name `STREAMS$_TRANSFORM_FUNCTION` in the name-value pair. The value in the name-value pair is the name of the PL/SQL procedure that performs the transformation.

The following query displays all of the rule-based transformations specified for rules in a rule set named `RULESET$_4`:

```
COLUMN RULE_NAME HEADING 'Rule Name' FORMAT A20
COLUMN ACTION_CONTEXT_VALUE HEADING 'Transformation Procedure' FORMAT A40

SELECT
    r.RULE_NAME,
    ac.NVN_VALUE.ACCESSVARCHAR2() ACTION_CONTEXT_VALUE
FROM DBA_RULES r,
     TABLE(R.RULE_ACTION_CONTEXT.ACTX_LIST) ac,
     DBA_RULE_SET_RULES s
WHERE ac.NVN_NAME      = 'STREAMS$_TRANSFORM_FUNCTION' AND
      s.RULE_SET_NAME  = 'RULESET$_4' AND
      s.RULE_SET_OWNER = 'STRMADMIN' AND
      r.RULE_NAME      = s.RULE_NAME AND
      r.RULE_OWNER     = s.RULE_OWNER;
```

If there rule-based transformations specified for rules in the rule set, then your output looks similar to the following:

Rule Name	Transformation Procedure
DEPARTMENTS7	hr.executive_to_management
DEPARTMENTS6	hr.executive_to_management
DEPARTMENTS5	hr.executive_to_management

See Also:

- ["Rule-Based Transformations"](#) on page 6-25
- ["Managing Rule-Based Transformations"](#) on page 15-11

Monitoring Streams Tags

The following sections contain queries that you can run to display the Streams tag for the current session and for an apply process:

- [Displaying the Tag Value for the Current Session](#)
- [Displaying the Tag Value for an Apply Process](#)

See Also:

- [Chapter 8, "Streams Tags"](#)
- ["Managing Streams Tags"](#) on page 16-26
- *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about the DBMS_STREAMS package

Displaying the Tag Value for the Current Session

You can display the tag value generated in all redo entries for the current session by querying the DUAL view:

```
SELECT DBMS_STREAMS.GET_TAG FROM DUAL;
```

Your output looks similar to the following:

```
GET_TAG
```

```
-----  
1D
```

You can also determine the tag for a session by calling the `DBMS_STREAMS.GET_TAG` function.

Displaying the Tag Value for an Apply Process

You can get the default tag for all redo entries generated by an apply process by querying for the `APPLY_TAG` value in the `DBA_APPLY` data dictionary view. For example, to get the hexadecimal value of the tags generated in the redo entries by an apply process named `strm01_apply`, run the following query:

```
SELECT APPLY_TAG "Tag Value for strm01_apply"
       FROM DBA_APPLY WHERE APPLY_NAME = 'STRM01_APPLY';
```

Your output looks similar to the following:

```
Tag Value for strm01_apply
```

```
-----
```

```
00
```

A handler or transformation function associated with an apply process can get the tag by calling the `DBMS_STREAMS.GET_TAG` function.

Troubleshooting a Streams Environment

This chapter contains information about identifying and resolving common problems in a Streams environment.

This chapter contains these topics:

- [Troubleshooting Capture Problems](#)
- [Troubleshooting Propagation Problems](#)
- [Troubleshooting Apply Problems](#)
- [Troubleshooting Problems with Rules and Rule-Based Transformations](#)
- [Checking the Trace Files and Alert Log for Problems](#)

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?](#)
- [Is LOG_PARALLELISM Set to 1?](#)
- [Is LOGMNR_MAX_PERSISTENT_SESSIONS Set High Enough?](#)

See Also:

- [Chapter 2, "Streams Capture Process"](#)
- [Chapter 12, "Managing a Capture Process"](#)
- ["Monitoring a Streams Capture Process" on page 17-3](#)

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 or aborted, then try restarting it. If you do not know why the capture process was disabled or aborted, then check the trace file for the capture process.

See Also:

- ["Starting a Capture Process"](#) on page 12-5
- ["Checking the Trace Files and Alert Log for Problems"](#) on page 18-24
- ["Streams Capture Processes and Oracle Real Application Clusters"](#) on page 2-17 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 a Capture Process"](#) on page 17-7
- ["Determining Event Enqueuing Latency for a Capture Process"](#) on page 17-8
- ["Capture Process Parallelism"](#) on page 2-26
- ["Setting a Capture Process Parameter"](#) on page 12-8

Is LOG_PARALLELISM Set to 1?

The `LOG_PARALLELISM` initialization parameter specifies the level of concurrency for redo allocation within Oracle. If you plan to run one or more capture processes on a database, then this parameter must be set to 1. If this initialization parameter is set higher than one, you may encounter error `ORA-01374`.

Setting this parameter to 1 does not affect the parallelism of capture. You can set parallelism for a capture process using the `SET_PARAMETER` procedure in the `DBMS_CAPTURE_ADM` package.

Is LOGMNR_MAX_PERSISTENT_SESSIONS Set High Enough?

The LOGMNR_MAX_PERSISTENT_SESSIONS initialization parameter specifies the maximum number of persistent LogMiner mining sessions that are concurrently active when all sessions are mining redo logs generated by instances. If you plan to run multiple Streams capture processes on a single database, then set this parameter equal to or higher than the number of planned capture processes.

If you cannot drop a capture process, and you are using multiple capture processes, then it may be because the LOGMNR_MAX_PERSISTENT_SESSIONS initialization parameter is not set high enough. Try increasing this initialization parameter and retrying the drop capture process operation.

Alternatively, if you do not want to increase the size of this initialization parameter, try stopping at least one of the running capture processes and then retrying the drop capture process operation. If the drop operation succeeds, then restart any capture process you stopped.

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 Streams Queue?](#)

See Also:

- [Chapter 3, "Streams Staging and Propagation"](#)
- [Chapter 13, "Managing Staging and Propagation"](#)
- ["Monitoring Streams Propagations and Propagation Jobs" on page 17-15](#)

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 `db1_to_db2`, 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.PROPGATION_NAME = 'DB1_TO_DB2';
```

Your output looks similar to the following:

Source Queue	Destination Queue
-----	-----
STRMADMIN.STREAMS_QUEUE@DB1.NET	STRMADMIN.STREAMS_QUEUE@DB2.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.

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 `db1_to_db2` 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 DB Link	Schedule	Process	Number of Failures	Last Error	Time Last Error	Message
DBS2.NET	Enabled	J001	0			

Try the following actions to correct a problem:

- 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 uncanceling and canceling the propagation job.

See Also:

- ["Enabling a Propagation Job"](#) on page 13-11
- ["Checking the Trace Files and Alert Log for Problems"](#) on page 18-24
- ["Uncanceling a Propagation Job"](#) on page 13-13
- ["Canceling a Propagation Job"](#) on page 13-11
- ["Displaying the Schedule for a Propagation Job"](#) on page 17-17

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 does propagation. 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 11-4](#)
- The description of propagation features in *Oracle9i Application Developer's Guide - Advanced Queuing* for more information about setting the `JOB_QUEUE_PROCESSES` initialization parameter when you use propagation jobs
- *Oracle9i Database Reference* for more information about the `JOB_QUEUE_PROCESSES` initialization parameter
- *Oracle9i Database Administrator's Guide* for more information about job queues

Is Security Configured Properly for the Streams Queue?

Streams queues are secure queues, and security must be configured properly for users to be able to perform operations on them. You may encounter one of the following errors if security is not configured properly for a Streams 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 3-22](#)

ORA-24093 AQ Agent not granted privileges of database user

Secure queue access must be granted to an 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 13-3 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 agent with secure queue privileges for the queue in the message properties.

See Also: ["Create the Procedure to Enqueue Non-LCR Events"](#) on page 19-12 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 a Custom Apply Handler Specified?](#)
- [Is the Apply Process Waiting for a Dependent Transaction?](#)
- [Are There Any Apply Errors in the Exception Queue?](#)

See Also:

- [Chapter 4, "Streams Apply Process"](#)
- [Chapter 14, "Managing an Apply Process"](#)
- ["Monitoring a Streams Apply Process"](#) on page 17-20

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 or aborted, then try restarting it. If you do not know why the apply process was disabled or aborted, then check the trace file for the apply process.

See Also:

- ["Starting an Apply Process"](#) on page 14-7
- ["Checking the Trace Files and Alert Log for Problems"](#) on page 18-24
- ["Streams Apply Processes and Oracle Real Application Clusters"](#) on page 4-29 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"](#) on page 17-31
- ["Apply Process Parallelism"](#) on page 4-34
- ["Setting an Apply Process Parameter"](#) on page 14-11

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 12-2

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, 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"](#) on page 4-4 for general information about apply handlers
- [Chapter 14, "Managing an Apply Process"](#) for information about managing apply handlers
- ["Displaying Information About Apply Handlers"](#) on page 17-23 for queries that display information about apply handlers

Is the Apply Process Waiting for a Dependent Transaction?

If you set the `parallelism` parameter for an apply process to a value greater than 1 and you set the `commit_serialization` parameter of the apply process to `full`, then the apply process may detect interested transaction list (ITL) contention if there is a transaction that is dependent on another transaction with a higher SCN. ITL contention occurs if the session that created the transaction waited for an ITL slot in a block. This happens when the session wants to lock a row in the block but one or more other sessions have rows locked in the same block, and there is no free ITL slot in the block.

ITL contention is also possible if the session is waiting due to shared bitmap index fragment. Bitmap indexes index key values and a range of rowids. Each 'entry' in a bitmap index can cover many rows in the actual table. If two sessions want to update rows covered by the same bitmap index fragment, then the second session waits for the first transaction to either `COMMIT` or `ROLLBACK`.

When an apply process detects such a dependency, it resolves the ITL contention automatically and records information about it in the alert log and apply process trace file for the database. ITL contention may negatively affect the performance of an apply process because there may not be any progress while it is detecting the deadlock.

To avoid the problem in the future, perform one of the following actions:

- Increase the number of ITLs available. You can do so by changing the `INITRANS` or `MAXTRANS` settings for the table using the `ALTER TABLE` statement.
- Set the `commit_serialization` parameter to `none` for the apply process.
- Set the `parallelism apply process` parameter to `1` for the apply process.

See Also:

- ["Apply Process Parameters"](#) on page 4-34
- ["Checking the Trace Files and Alert Log for Problems"](#) on page 18-24
- *Oracle9i Database Administrator's Guide* and *Oracle9i SQL Reference* for more information about `INITRANS` and `MAXTRANS`

Are There Any Apply Errors in the Exception Queue?

When an apply process cannot apply an event, it moves the event and all of the other events in the same transaction into its queue's exception queue. You should check the 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.

See Also:

- ["Checking for Apply Errors"](#) on page 17-36
- ["Managing Apply Errors"](#) on page 14-33

You may encounter the following types of apply process errors for LCR events:

- [ORA-01403 No Data Found](#)
- [ORA-26687 Instantiation SCN Not Set](#)
- [ORA-26688 Metadata Mismatch](#)
- [ORA-26689 Column Type Mismatch](#)

ORA-01403 No Data Found

Typically, this error occurs when an update is attempted on an existing row and the `OLD_VALUES` in the row LCR do not match the current values at this target site.

To correct this problem, you can update the current values in the row so that the row LCR can be applied successfully. If changes to the row are captured by a capture process at the destination database, then you probably do not want apply this manual change at destination sites. In this case, complete the following steps:

1. Set an apply tag in the session that corrects the row. Make sure you set the tag to a value that prevents the manual change from being applied at some of all destination databases.

```
EXEC DBMS_STREAMS.SET_TAG(tag => HEXTORAW('17'));
```

2. Update the row to correct the old values.

3. Reexecute the error or reexecute all errors. To reexecute an error, run the `EXECUTE_ERROR` procedure in the `DBMS_APPLY_ADM` package, and specify the transaction identifier for the transaction that caused the error.

```
EXEC DBMS_APPLY_ADM.EXECUTE_ERROR(local_transaction_id => '5.4.312');
```

Or, execute all errors for the apply process by running the `EXECUTE_ALL_ERRORS` procedure:

```
EXEC DBMS_APPLY_ADM.EXECUTE_ALL_ERRORS(apply_name => 'APPLY');
```

4. If you are going to make other changes in the current session that you want to apply at destination databases, then reset the tag for the session to an appropriate value, as in the following example:

```
EXEC DBMS_STREAMS.SET_TAG(tag => NULL);
```

ORA-26687 Instantiation SCN Not Set

Typically, this error occurs because the instantiation SCN is not set on an object for which an apply process is attempting to apply changes. You can query the `DBA_APPLY_INSTANTIATED_OBJECTS` data dictionary view to list the objects that have an instantiation SCN.

You can set an instantiation SCN for one or more objects by exporting the objects at the source database, and then importing them at the destination database. If you do not want to use Export/Import, then you can run one or more of the following procedures in the `DBMS_APPLY_ADM` package:

- `SET_TABLE_INSTANTIATION_SCN`
- `SET_SCHEMA_INSTANTIATION_SCN`
- `SET_GLOBAL_INSTANTIATION_SCN`

See Also: ["Setting Instantiation SCNs at a Destination Database"](#) on page 14-35

ORA-26688 Metadata Mismatch

Typically, this error occurs because of one of the following conditions:

- The object for which an LCR is applying a change does not exist in the destination database. In this case, check to see if the object exists. Also, make sure you use the correct character case in rule conditions and apply handlers. For example, if a column name has all uppercase characters in the data dictionary, then you should specify the column name with all uppercase characters in rule conditions and in apply handlers.
- There is a problem with the primary key in the table for which an LCR is applying a change. In this case, make sure the primary key is enabled by querying the `DBA_CONSTRAINTS` data dictionary view. If no primary key exists for the table, then specify substitute key columns using the `SET_KEY_COLUMNS` procedure in the `DBMS_APPLY_ADM` package. You may also encounter error `ORA-23416` if a table being applied does not have a primary key.

See Also:

- ["Considerations for Applying DML Changes to Tables"](#) on page 4-10
- ["Managing the Substitute Key Columns for a Table"](#) on page 14-27

- Supplemental logging is not specified for columns that require supplemental logging at the source database. In this case, LCRs from the source database may not contain values for key columns.

See Also:

- ["Displaying Supplemental Log Groups at a Source Database"](#) on page 17-11
- ["Supplemental Logging in a Streams Environment"](#) on page 2-11
- ["Specifying Supplemental Logging at a Source Database"](#) on page 12-9

ORA-26689 Column Type Mismatch

Typically, this error occurs because one or more columns at a table in the source database do not match the corresponding columns at the destination database. The LCRs from the source database may contain more columns than the table at the destination database, or there may be a type mismatch for one or more columns. If the columns differ at the databases, you can use rule-based transformations to avoid errors.

This error may also occur because supplemental logging is not specified where it is required for nonkey columns at the source database. In this case, LCRs from the source database may not contain needed values for these nonkey columns.

See Also:

- ["Considerations for Applying DML Changes to Tables"](#) on page 4-10
- [Rule-Based Transformations](#) on page 6-25
- ["Managing Rule-Based Transformations"](#) on page 15-11
- ["Supplemental Logging in a Streams Environment"](#) on page 2-11

Troubleshooting Problems with Rules and Rule-Based Transformations

If a capture process, a propagation, or an apply process is not behaving as expected, then the problem may be that rules or rule-based transformations for the capture process, propagation, or apply process 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 Process or Propagation?](#)
- [Are the Rule-Based Transformations Configured Properly?](#)

Note:

- [Chapter 5, "Rules"](#)
- [Chapter 6, "How Rules Are Used In Streams"](#)
- [Chapter 15, "Managing Rules and Rule-Based Transformations"](#)

Are Rules Configured Properly for the Streams Process or Propagation?

If a Streams capture process, propagation, or apply process is behaving in an unexpected way, then the problem may be that the rules in the rule set for the capture process, propagation, or apply process 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 rule set for the capture process does not contain a rule that evaluates to `TRUE` when a change is made to the object.

You can check the rules for a particular Streams capture process, propagation, or apply process by querying the following data dictionary views:

- `DBA_STREAMS_TABLE_RULES`
- `DBA_STREAMS_SCHEMA_RULES`
- `DBA_STREAMS_GLOBAL_RULES`

Note: These data dictionary views contain information about rules created using the `DBMS_STREAMS_ADM` package or the Streams tool in Oracle Enterprise Manager. They do not contain information about rules created using the `DBMS_RULE_ADM` package. To view information about rules created with the `DBMS_RULE_ADM` package, query the `DBA_RULES` data dictionary view.

See Also: ["Monitoring Rules and Rule-Based Transformations"](#) on page 17-42

A rule set with no rules is not the same as no rule set. For example, if you use a rule set with no rules for a propagation, then the propagation will not propagate anything. If you do not use a rule set at all for a propagation, then the propagation propagates everything in its source queue.

This section includes the following subsections:

- [Example That Checks for Capture Process Rules on a Table](#)
- [Example That Checks for Apply Process Rules on a Table](#)
- [Checking for Schema and Global Rules](#)
- [Resolving Problems with Rules](#)

Example That Checks for Capture Process Rules on a Table

For example, suppose a database is running a capture process named `CAPTURE`, and you want to check for table rules that evaluate to `TRUE` for this capture process when there are changes to the `hr.departments` table. To determine whether there are any such rules in the rule set for the capture process, run the following query:

```
COLUMN RULE_NAME HEADING 'Rule Name' FORMAT A30
COLUMN RULE_TYPE HEADING 'Rule Type' FORMAT A30

SELECT RULE_NAME, RULE_TYPE
FROM DBA_STREAMS_TABLE_RULES
WHERE STREAMS_NAME = 'CAPTURE' AND
      STREAMS_TYPE = 'CAPTURE' AND
      TABLE_OWNER = 'HR' AND
      TABLE_NAME = 'DEPARTMENTS';
```

Your output looks similar to the following:

Rule Name	Rule Type
DEPARTMENTS1	DML

Based on these results, the capture process named `CAPTURE` should capture DML changes to the `hr.departments` table. In other words, a rule exists in the capture process rule set that evaluates to `TRUE` when the capture process finds a DML change to the `hr.departments` table in the redo log.

A rule of type `DDL` for the table in the query results means that the capture process should capture DDL changes to the table. If a capture process should capture both DML and DDL changes to the table, then a rule of each type would appear for the table in the query results.

Example That Checks for Apply Process Rules on a Table

If you expect an apply process to apply changes to a particular table, but the apply process is not applying these changes, then the cause may be that the rule set for the apply process does not contain a rule that evaluates to `TRUE` when an LCR is in the apply process queue. As with capture rules and propagation rules, you can check the rules that were created using the procedures in the `DBMS_STREAMS_ADM` package by querying the following data dictionary views:

- `DBA_STREAMS_TABLE_RULES`
- `DBA_STREAMS_SCHEMA_RULES`
- `DBA_STREAMS_GLOBAL_RULES`

In addition, 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 rule set for the capture process that captures these events must be configured properly. Similarly, if events are propagated from one or more databases before reaching the apply process, then the rules for each propagation must be configured properly. If the rules for a capture process or a propagation on which the apply process depends are not configured properly, then the events may never reach the apply process queue.

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 the capture process or propagations on which an apply process depends, or a problem is caused by the apply process itself:

- If no other destination databases of the 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 rules for the capture process are configured properly, and then check the rules for the propagations nearest the capture process.
- If other destination databases of the 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 rules for the apply process are configured properly, and then check the rules for the propagations nearest the apply process.

Also, when you are checking for apply rules, there is the possibility that subset rules exist for one or more tables. A subset rules evaluates to `TRUE` only if an LCR 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 `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_TABLE_RULES
WHERE STREAMS_NAME = 'APPLY' AND
      STREAMS_TYPE = 'APPLY' AND
      TABLE_OWNER = 'HR' AND
      TABLE_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. 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 the subset rules cause the apply process to discard changes that involve rows where the `location_id` is not 1700.

See Also:

- ["Row Subsetting Using Streams Rules"](#) on page 4-12 for conceptual information about subset rules
- ["Table-Level Rules Example"](#) on page 6-7 for more information about specifying subset rules

Checking for Schema and Global Rules

Schema rules or global rules may also be used to capture changes to all of the database objects in a particular schema or database, respectively. For example, to check for schema rules that evaluate to `TRUE` for a capture process named `CAPTURE` when there are changes to the `hr` schema, run the following query:

```
COLUMN RULE_NAME HEADING 'Rule Name' FORMAT A20
COLUMN RULE_TYPE HEADING 'Rule Type' FORMAT A20

SELECT RULE_NAME, RULE_TYPE
       FROM DBA_STREAMS_SCHEMA_RULES
       WHERE STREAMS_NAME = 'CAPTURE' AND
             SCHEMA_NAME  = 'HR' ;
```

Your output looks similar to the following:

Rule Name	Rule Type
HR7	DML
HR8	DDL

Based on these results, the capture process named `CAPTURE` should capture DML and DDL changes to all objects in the `hr` schema.

If the `DBA_STREAMS_GLOBAL_RULES` data dictionary view returns any rows when you query it for a capture process, then the capture process captures all changes in the database, except for unsupported changes and changes made to the `SYS` and `SYSTEM` schemas.

See Also: ["Datatypes Captured"](#) on page 2-6 and ["Types of Changes Captured"](#) on page 2-7 for information about the types of changes that are captured and are not captured by a capture process

Resolving Problems with Rules

If you determine that a Streams capture process, propagation, or apply process is not behaving as expected because one or more rules must be added to the rule set for the capture process, propagation, or apply process, 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_RULES`
- `ADD_TABLE_PROPAGATION_RULES`
- `ADD_TABLE_RULES`

You can use the `DBMS_RULE_ADM` package to add customized rules, if necessary.

It is also possible that the Streams capture process, propagation, or apply process 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 changes for the relevant objects 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 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 rules or reusing existing capture 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`

See Also:

- ["Altering a Rule"](#) on page 15-5
- ["Removing a Rule from a Rule Set"](#) on page 15-7
- ["Preparing Database Objects for Instantiation at a Source Database"](#) on page 12-11
- ["Data Dictionary Duplication During Capture Process Creation"](#) on page 2-22 for more information about the Streams data dictionary

Are the Rule-Based Transformations Configured Properly?

A rule-based transformation is any modification to an event that results when a rule 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 procedure for the value. This user-created procedure performs the transformation. If the user-defined procedure contains any flaws, then unexpected behavior may result.

If a Streams capture process, propagation, or apply process is not behaving as expected, then check the rule-based transformation procedures specified for the capture process, propagation, or apply process, and correct any flaws. You can find the names of these procedures by querying the action context of the rules in the rule set used by the Streams capture process, propagation, or apply process. You may need to modify a transformation procedure or remove a rule-based transformation to correct the problem. Make sure the name portion of the name-value pair in the action context 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 Set, and Status of Each Capture Process"](#) on page 17-3 for a query that displays the rule set used by a capture process
- ["Displaying the Rule-Based Transformations in a Rule Set"](#) on page 17-49 for a query that displays the rule-based transformation procedures specified for the rules in a rule set
- ["Managing Rule-Based Transformations"](#) on page 15-11 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 capture process runs on a source 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_XXXXXX_IIIII.trc`, where:

- `sid` is the system identifier for the database
- `XXXXXX` 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 the capture process and 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_file` 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:

- *Oracle9i Database Administrator's Guide* for more information about trace files and the alert log, and for more information about their names and locations
- *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about setting the `trace_file` capture process parameter and the `trace_file` 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 `cpnn`, where `nn` 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_cp01`.

See Also: ["Displaying General Information About a Capture Process"](#) on page 17-4 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 `cjqnn`, where `nn` is the job queue coordinator process number, and a job queue process is named `jnnn`, 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 18-6 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 `apnn`, where `nn` 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 `01`, then the trace file for the apply process starts with `hqdb_ap01`.

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 `pnnn`, 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 an apply process starts with `hqdb_p001`.

See Also:

- ["Displaying Information About the Coordinator Process"](#) on page 17-30 for a query that displays the apply process number of an apply process
- ["Displaying Information About the Reader Server for an Apply Process"](#) on page 17-28 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 an Apply Process"](#) on page 17-30 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:

- [Chapter 19, "Streams Messaging Example"](#)
- [Chapter 20, "Single Database Capture and Apply Example"](#)
- [Chapter 21, "Simple Single Source Replication Example"](#)
- [Chapter 22, "Single Source Heterogeneous Replication Example"](#)
- [Chapter 23, "Multiple Source Replication Example"](#)
- [Chapter 24, "Rule-Based Application Example"](#)

Streams Messaging Example

This chapter illustrates a messaging environment that can be constructed using Streams.

This chapter contains these topics:

- [Overview of Messaging Example](#)
- [Prerequisites](#)
- [Set Up Users and Create a Streams Queue](#)
- [Create the Enqueue Procedures](#)
- [Configure an Apply Process](#)
- [Configure Explicit Dequeue](#)
- [Enqueue Events](#)
- [Dequeue Events Explicitly and Query for Applied Events](#)
- [Enqueue and Dequeue Events Using JMS](#)

See Also:

- [Chapter 3, "Streams Staging and Propagation"](#)
- [Chapter 13, "Managing Staging and Propagation"](#)
- ["Monitoring a Streams Queue" on page 17-12](#)

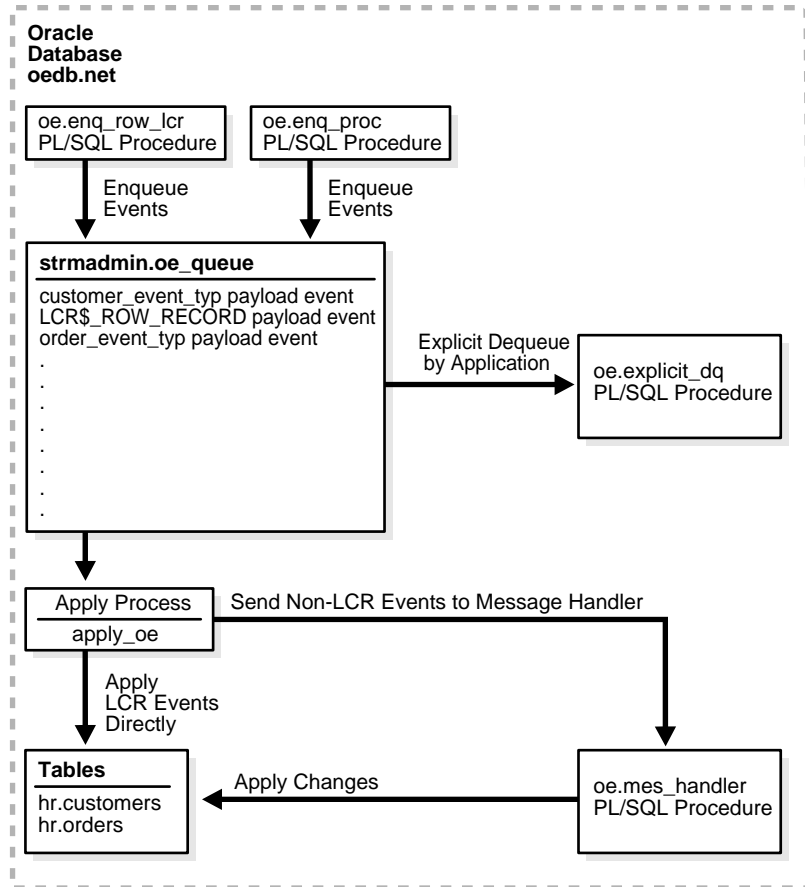
Overview of Messaging Example

This example illustrates using a single `SYS.AnyData` queue at a database called `oedb.net` to create a Streams messaging environment in which events containing message payloads of different types are stored in the same queue. Specifically, this example illustrates the following messaging features of Streams:

- Enqueuing messages containing order payload and customer payload as `SYS.Anydata` events into the queue
- Enqueuing messages containing row LCR payload as `SYS.Anydata` events into the queue
- Creating a rule set for applying the events
- Creating an evaluation context used by the rule set
- Creating a Streams apply process to dequeue and process the events based on rules
- Creating a message handler and associating it with the apply process
- Explicitly dequeuing and processing events based on rules without using the apply process

Figure 19-1 provides an overview of this environment.

Figure 19-1 Example Streams Messaging Environment



Prerequisites

The following are prerequisites that must be completed before you begin the example in this section.

- Set the following initialization parameters to the values indicated for all databases in the environment:
 - `AQ_TM_PROCESSES`: This parameter establishes queue monitor processes. Values from 1 to 10 specify the number of queue monitor processes created to monitor the messages. If `AQ_TM_PROCESSES` is not specified or is set to 0, then the queue monitor processes are not created. In this example, `AQ_TM_PROCESSES` should be set to at least 1.

Setting the parameter to 1 or more starts the specified number of queue monitor processes. These queue monitor processes are responsible for managing time-based operations of messages such as delay and expiration, cleaning up retained messages after the specified retention time, and cleaning up consumed messages if the retention time is 0.
 - `COMPATIBLE`: This parameter must be set to 9.2.0 or higher.
- Configure your network and Oracle Net so that you can access the `oedb.net` database from the client where you run these scripts.

See Also: *Oracle9i Net Services Administrator's Guide*

- This examples 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.

Set Up Users and Create a Streams Queue

Complete the following steps to set up users and create a Streams queue for a Streams messaging environment.

1. [Show Output and Spool Results](#)
2. [Set Up Users](#)
3. [Create the Streams Queue](#)
4. [Check the Spool Results](#)

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the next "END OF SCRIPT" line on page 19-9 into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to the database.

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

Step 1 Show Output and Spool Results

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

```
*/  
  
SET ECHO ON  
SPOOL streams_setup_message.out  
  
/*
```

Step 2 Set Up Users

Connect to `oedb.net` as `SYS` user.

```
*/  
  
CONNECT SYS/CHANGE_ON_INSTALL@oedb.net AS SYSDBA  
  
/*
```

This example uses the `oe` sample schema. For this example to work properly, the `oe` user must have privileges to execute the subprograms in the `DBMS_AQ` package. The `oe` user will be specified as the queue user when the Streams queue is created in Step 3. The `SET_UP_QUEUE` procedure will grant the `oe` user `ENQUEUE` and `DEQUEUE` privileges on the queue, but the `oe` user also needs `EXECUTE` privilege on the `DBMS_AQ` package to enqueue events into and dequeue events from the queue.

Also, most of the configuration and administration actions illustrated in this example are performed by the Streams administrator. In this step, create the Streams administrator named `strmadmin` and grant this user the necessary privileges. These privileges enable the user to execute subprograms in packages related to Streams, create rule sets, create rules, and monitor the Streams environment by querying data dictionary views. You may choose a different name for this user.

Note:

- To ensure security, use a password other than `strmadminpw` for the Streams administrator.
 - The `SELECT_CATALOG_ROLE` is not required for the Streams administrator. It is granted in this example so that the Streams administrator can monitor the environment easily.
 - If you plan to use the Streams tool in Oracle Enterprise Manager, then grant the Streams administrator `SELECT ANY DICTIONARY` privilege, in addition to the privileges shown in this step.
 - The `ACCEPT` command must appear on a single line in the script.
-
-

```
*/

GRANT EXECUTE ON DBMS_AQ TO oe;

GRANT CONNECT, RESOURCE, SELECT_CATALOG_ROLE
  TO strmadmin IDENTIFIED BY strmadminpw;

ACCEPT streams_tbs PROMPT 'Enter the tablespace for the Streams administrator: '

ALTER USER strmadmin DEFAULT TABLESPACE &streams_tbs
  QUOTA UNLIMITED ON &streams_tbs;
```

```

GRANT EXECUTE ON DBMS_APPLY_ADM TO strmadmin;
GRANT EXECUTE ON DBMS_AQ TO strmadmin;
GRANT EXECUTE ON DBMS_AQADM TO strmadmin;
GRANT EXECUTE ON DBMS_STREAMS_ADM TO strmadmin;

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

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

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

/*

```

Step 3 Create the Streams Queue

Connect as the Streams administrator.

```

*/

CONNECT strmadmin/strmadminpw@oedb.net

/*

```

Run the `SET_UP_QUEUE` procedure to create a queue named `oe_queue` at `oedb.net`. This queue will function as the Streams queue by holding events used in the messaging environment.

Running the `SET_UP_QUEUE` procedure performs the following actions:

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

```
*/  
  
BEGIN  
  DBMS_STREAMS_ADM.SET_UP_QUEUE(  
    queue_table => 'oe_queue_table',  
    queue_name  => 'oe_queue');  
END;  
/  
  
/*
```

Step 4 Grant the oe User Privileges on the Queue

```
*/  
  
BEGIN  
  SYS.DBMS_AQADM.GRANT_QUEUE_PRIVILEGE(  
    privilege => 'ALL',  
    queue_name => 'strmadmin.oe_queue',  
    grantee   => 'oe');  
END;  
/  
  
/*
```

Step 5 Create an Agent for Explicit Enqueue

Create an agent that will be used to perform explicit enqueue operations on the `oe_queue` queue.

```
*/

BEGIN
  SYS.DBMS_AQADM.CREATE_AQ_AGENT(
    agent_name => 'explicit_enq');
END;
/

/*
```

Step 6 Associate the oe User with the explicit_enq Agent

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. The `oe_queue` queue is a secure queue because it was created using `SET_UP_QUEUE`. This step enables the `oe` user to perform enqueue operations on this queue.

```
*/

BEGIN
  DBMS_AQADM.ENABLE_DB_ACCESS(
    agent_name => 'explicit_enq',
    db_username => 'oe');
END;
/

/*
```

Step 7 Check the Spool Results

Check the `streams_setup_message.out` spool file to ensure that all actions completed successfully after this script completes.

```
*/

SET ECHO OFF
SPOOL OFF

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

Create the Enqueue Procedures

Complete the following steps to create one PL/SQL procedure that enqueues non-LCR events into the Streams queue and one PL/SQL procedure that enqueues row LCR events into the Streams queue.

1. [Show Output and Spool Results](#)
2. [Create a Type to Represent Orders](#)
3. [Create a Type to Represent Customers](#)
4. [Create the Procedure to Enqueue Non-LCR Events](#)
5. [Create a Procedure to Construct and Enqueue Row LCR Events](#)
6. [Check the Spool Results](#)

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the next "END OF SCRIPT" line on page 19-14 into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to the database.

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

Step 1 Show Output and Spool Results

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

```
*/
```

```
SET ECHO ON  
SPOOL streams_enqprocs_message.out
```

```
/*
```

Step 2 Create a Type to Represent Orders

Connect as oe.

```
*/
```

```
CONNECT oe/oe@oedb.net
```

```
/*
```

Create a type to represent orders based on the columns in the `oe.orders` table. The type attributes include the columns in the `oe.orders` table, along with one extra attribute named `action`. The value of the `action` attribute for instances of this type will be used to determine correct action to perform on the instance (either apply process dequeue or explicit dequeue). This type will be used for events that will be enqueued into the Streams queue.

```
*/
```

```
CREATE OR REPLACE TYPE order_event_typ AS OBJECT (  
    order_id      NUMBER(12),  
    order_date    TIMESTAMP(6) WITH LOCAL TIME ZONE,  
    order_mode    VARCHAR2(8),  
    customer_id   NUMBER(6),  
    order_status  NUMBER(2),  
    order_total   NUMBER(8,2),  
    sales_rep_id  NUMBER(6),  
    promotion_id  NUMBER(6),  
    action        VARCHAR(7));
```

```
/
```

```
/*
```

Step 3 Create a Type to Represent Customers

Create a type to represent customers based on the columns in the `oe.customers` table. The type attributes include the columns in the `oe.customers` table, along with one extra attribute named `action`. The value of the `action` attribute for instances of this type will be used to determine correct action to perform on the instance (either apply process dequeue or explicit dequeue). This type will be used for events that will be enqueued into the Streams queue.

```
*/  
  
CREATE OR REPLACE TYPE customer_event_typ AS OBJECT (  
    customer_id          NUMBER(6),  
    cust_first_name      VARCHAR2(20),  
    cust_last_name       VARCHAR2(20),  
    cust_address         CUST_ADDRESS_TYP,  
    phone_numbers        PHONE_LIST_TYP,  
    nls_language         VARCHAR2(3),  
    nls_territory        VARCHAR2(30),  
    credit_limit         NUMBER(9,2),  
    cust_email           VARCHAR2(30),  
    account_mgr_id       NUMBER(6),  
    cust_geo_location    MDSYS.SDO_GEOMETRY,  
    action               VARCHAR(7));  
/  
  
/*
```

Step 4 Create the Procedure to Enqueue Non-LCR Events

Create a PL/SQL procedure called `enq_proc` to enqueue events into the Streams queue.

Note: A single enqueued message can be dequeued by an apply process and by an explicit dequeue, but this example does not illustrate this capability.

```

*/
CREATE OR REPLACE PROCEDURE oe.enq_proc (event IN SYS.Anydata) IS
    enqopt      DBMS_AQ.ENQUEUE_OPTIONS_T;
    mprop       DBMS_AQ.MESSAGE_PROPERTIES_T;
    enq_eventid RAW(16);
BEGIN
    mprop.SENDER_ID := SYS.AQ$_AGENT('explicit_enq', NULL, NULL);
    DBMS_AQ.ENQUEUE(
        queue_name      => 'strmadmin.oe_queue',
        enqueue_options => enqopt,
        message_properties => mprop,
        payload          => event,
        msgid            => enq_eventid);
END;
/
/*

```

Step 5 Create a Procedure to Construct and Enqueue Row LCR Events

Create a procedure called `enq_row_lcr` that constructs a row LCR and then enqueues the row LCR into the queue.

See Also: *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about LCR constructors

```

*/

CREATE OR REPLACE PROCEDURE oe.enq_row_lcr(
            source_dbname  VARCHAR2,
            cmd_type       VARCHAR2,
            obj_owner      VARCHAR2,
            obj_name       VARCHAR2,
            old_vals       SYS.LCR$_ROW_LIST,
            new_vals       SYS.LCR$_ROW_LIST) AS
    eopt          DBMS_AQ.ENQUEUE_OPTIONS_T;
    mprop        DBMS_AQ.MESSAGE_PROPERTIES_T;
    enq_msgid    RAW(16);
    row_lcr      SYS.LCR$_ROW_RECORD;
BEGIN
    mprop.SENDER_ID := SYS.AQ$_AGENT('explicit_enq', NULL, NULL);
    -- Construct the LCR based on information passed to procedure
    row_lcr := SYS.LCR$_ROW_RECORD.CONSTRUCT(
        source_database_name => source_dbname,
        command_type         => cmd_type,
        object_owner         => obj_owner,
        object_name          => obj_name,
        old_values            => old_vals,
        new_values            => new_vals);
    -- Enqueue the created row LCR
    DBMS_AQ.ENQUEUE(
        queue_name           => 'strmadmin.oe_queue',
        enqueue_options      => eopt,
        message_properties   => mprop,
        payload               => SYS.AnyData.ConvertObject(row_lcr),
        msgid                 => enq_msgid);
END enq_row_lcr;
/

/*

```

Step 6 Check the Spool Results

Check the `streams_enqprocs_message.out` spool file to ensure that all actions completed successfully after this script completes.

```

*/

SET ECHO OFF
SPOOL OFF

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

```

Configure an Apply Process

Complete the following steps to configure an apply process to apply the user-enqueued events in the Streams queue.

1. [Show Output and Spool Results](#)
2. [Create a Function to Determine the Value of the action Attribute](#)
3. [Create a Message Handler](#)
4. [Grant stradmin EXECUTE Privilege on the Procedures](#)
5. [Create the Evaluation Context for the Rule Set](#)
6. [Create a Rule Set for the Apply Process](#)
7. [Create a Rule that Evaluates to TRUE if the Event Action Is apply](#)
8. [Create a Rule that Evaluates to TRUE for the Row LCR Events](#)
9. [Add the Rules to the Rule Set](#)
10. [Create an Apply Process](#)
11. [Grant EXECUTE Privilege on the Rule Set To oe User](#)
12. [Start the Apply Process](#)
13. [Check the Spool Results](#)

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the next "END OF SCRIPT" line on page 19-23 into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to the database.

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

Step 1 Show Output and Spool Results

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

```
*/  
  
SET ECHO ON  
SPOOL streams_apply_message.out  
  
/*
```

Step 2 Create a Function to Determine the Value of the action Attribute

Connect as `oe`.

```
*/  
  
CONNECT oe/oe@oedb.net  
  
/*
```

Create a function called `get_oe_action` to determine the value of the `action` attribute in the events in the queue. This function is used in rules later in this example to determine the value of the `action` attribute for an event. Then, the clients of the rules engine perform the appropriate action for the event (either dequeue by apply process or explicit dequeue). In this example, the clients of the rules engine are the apply process and the `oe.explicit_dq` PL/SQL procedure.

```

*/

CREATE OR REPLACE FUNCTION oe.get_oe_action (event IN SYS.Anydata)
RETURN VARCHAR2
IS
  ord          oe.order_event_typ;
  cust         oe.customer_event_typ;
  num          NUMBER;
  type_name    VARCHAR2(61);
BEGIN
  type_name := event.GETTYPENAME;
  IF type_name = 'OE.ORDER_EVENT_TYP' THEN
    num := event.GETOBJECT(ord);
    RETURN ord.action;
  ELSIF type_name = 'OE.CUSTOMER_EVENT_TYP' THEN
    num := event.GETOBJECT(cust);
    RETURN cust.action;
  ELSE
    RETURN NULL;
  END IF;
END;
/

/*

```

Step 3 Create a Message Handler

Create a message handler called `mes_handler` that will be used as a message handler by the apply process. This procedure takes the payload in a user-enqueued event of type `oe.order_event_typ` or `oe.customer_event_typ` and inserts it as a row in the `oe.orders` table and `oe.customers` table, respectively.

```
*/  
  
CREATE OR REPLACE PROCEDURE oe.mes_handler (event SYS.AnyData)  
IS  
    ord            oe.order_event_typ;  
    cust           oe.customer_event_typ;  
    num            NUMBER;  
    type_name      VARCHAR2(61);  
BEGIN  
    type_name := event.GETTYPENAME;  
    IF type_name = 'OE.ORDER_EVENT_TYP' THEN  
        num := event.GETOBJECT(ord);  
        INSERT INTO oe.orders VALUES (ord.order_id, ord.order_date,  
            ord.order_mode, ord.customer_id, ord.order_status, ord.order_total,  
            ord.sales_rep_id, ord.promotion_id);  
    ELSIF type_name = 'OE.CUSTOMER_EVENT_TYP' THEN  
        num := event.GETOBJECT(cust);  
        INSERT INTO oe.customers VALUES (cust.customer_id, cust.cust_first_name,  
            cust.cust_last_name, cust.cust_address, cust.phone_numbers,  
            cust.nls_language, cust.nls_territory, cust.credit_limit, cust.cust_email,  
            cust.account_mgr_id, cust.cust_geo_location);  
    END IF;  
END;  
/  
  
/*
```

Step 4 Grant strmadmin EXECUTE Privilege on the Procedures

```
*/  
  
GRANT EXECUTE ON get_oe_action TO strmadmin;  
  
GRANT EXECUTE ON mes_handler TO strmadmin;  
  
/*
```

Step 5 Create the Evaluation Context for the Rule Set

Connect as the Streams administrator.

```
*/
CONNECT strmadmin/strmadminpw@oedb.net
/*
```

Create the evaluation context for the rule set. The table alias is `tab` in this example, but you can use a different table alias name if you wish.

```
*/
DECLARE
    table_alias      SYS.RE$TABLE_ALIAS_LIST;
BEGIN
    table_alias := SYS.RE$TABLE_ALIAS_LIST(SYS.RE$TABLE_ALIAS(
                                                'tab',
                                                'strmadmin.oe_queue_table'));
    DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT(
        evaluation_context_name => 'oe_eval_context',
        table_aliases           => table_alias);
END;
/
/*
```

Step 6 Create a Rule Set for the Apply Process

Create the rule set for the apply process.

```
*/
BEGIN
    DBMS_RULE_ADM.CREATE_RULE_SET(
        rule_set_name      => 'apply_oe_rs',
        evaluation_context => 'strmadmin.oe_eval_context');
END;
/
/*
```

Step 7 Create a Rule that Evaluates to TRUE if the Event Action Is apply

Create a rule that evaluates to TRUE if the action value of an event is apply. Notice that `tab.user_data` is passed to the `oe.get_oe_action` function. The `tab.user_data` column holds the event payload in a queue table. The table alias for the queue table was specified as `tab` in Step 5 on page 19-19.

```
*/

BEGIN
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name    => 'strmadmin.apply_action',
    condition    => ' oe.get_oe_action(tab.user_data) = ''APPLY'' ');
END;
/

/*
```

Step 8 Create a Rule that Evaluates to TRUE for the Row LCR Events

Create a rule that evaluates to TRUE if the event in the queue is a row LCR that changes either the `oe.orders` table or the `oe.customers` table. This rule will enable the apply process to apply user-enqueued changes to the tables directly. For convenience, this rule uses the Oracle-supplied evaluation context `SYS.STREAMS$_EVALUATION_CONTEXT` because the rule is used to evaluate LCRs. When this rule is added to the rule set, this evaluation context is used for the rule during evaluation instead of the rule set's evaluation context.

```
*/

BEGIN
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name      => 'apply_lcrs',
    condition      => ':dml.GET_OBJECT_OWNER() = ''OE'' AND ' ||
                    ' (:dml.GET_OBJECT_NAME() = ''ORDERS'' OR ' ||
                    ':dml.GET_OBJECT_NAME() = ''CUSTOMERS'') ',
    evaluation_context => 'SYS.STREAMS$_EVALUATION_CONTEXT');
END;
/

/*
```


Step 9 Add the Rules to the Rule Set

Add the rules created in Step 7 and Step 8 to the rule set created in Step 6 on page 19-19.

```

*/

BEGIN
  DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'apply_action',
    rule_set_name  => 'apply_oe_rs');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'apply_lcrs',
    rule_set_name  => 'apply_oe_rs');
END;
/

/*

```

Step 10 Create an Apply Process

Create an apply process that is associated with the `oe_queue`, that uses the `apply_oe_rs` rule set, and that uses the `mes_handler` procedure as a message handler.

```

*/

BEGIN
  DBMS_APPLY_ADM.CREATE_APPLY(
    queue_name      => 'strmadmin.oe_queue',
    apply_name      => 'apply_oe',
    rule_set_name   => 'strmadmin.apply_oe_rs',
    message_handler => 'oe.mes_handler',
    apply_user      => 'oe',
    apply_captured  => false);
END;
/

/*

```

Step 11 Grant EXECUTE Privilege on the Rule Set To oe User

Grant EXECUTE privilege on the `strmadmin.apply_oe_rs` rule set. Because `oe` was specified as the apply user when the apply process was created in Step 10, `oe` needs execute privilege on the rule set used by the apply process.

```

*/

BEGIN
  DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.EXECUTE_ON_RULE_SET,
    object_name  => 'strmadmin.apply_oe_rs',
    grantee     => 'oe',
    grant_option => FALSE);
END;
/

/*

```

Step 12 Start the Apply Process

Set the `disable_on_error` parameter to `n` so that the apply process is not disabled if it encounters an error, and start the apply process at `oedb.net`.

```

*/

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

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

/*

```

Step 13 Check the Spool Results

Check the `streams_apply_message.out` spool file to ensure that all actions completed successfully after this script completes.

```
*/

SET ECHO OFF
SPOOL OFF

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

Configure Explicit Dequeue

Complete the following steps to configure explicit dequeue of messages based on message contents.

1. [Show Output and Spool Results](#)
2. [Create an Agent for Explicit Dequeue](#)
3. [Associate the oe User with the explicit_dq Agent](#)
4. [Add a Subscriber to the oe_queue Queue](#)
5. [Create a Procedure to Dequeue Events Explicitly](#)
6. [Check Spool Results](#)

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the next "END OF SCRIPT" line on page 19-28 into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to the database.

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

Step 1 Show Output and Spool Results

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

```
*/  
  
SET ECHO ON  
SPOOL streams_explicit_dq.out  
  
/*
```

Step 2 Create an Agent for Explicit Dequeue

Connect as the Streams administrator.

```
*/  
  
CONNECT strmadmin/strmadminpw@oedb.net  
  
/*
```

Create an agent that will be used to perform explicit dequeue operations on the `oe_queue` queue.

```
*/  
  
BEGIN  
  SYS.DBMS_AQADM.CREATE_AQ_AGENT(  
    agent_name => 'explicit_dq');  
END;  
/  
  
/*
```

Step 3 Associate the `oe` User with the `explicit_dq` Agent

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. The `oe_queue` queue is a secure queue because it was created using `SET_UP_QUEUE`. The `oe` user will be able to perform dequeue operations on this queue when the agent is used to create a subscriber to the queue in the next step.

```

*/
BEGIN
  DBMS_AQADM.ENABLE_DB_ACCESS(
    agent_name => 'explicit_dq',
    db_username => 'oe');
END;
/

/*

```

Step 4 Add a Subscriber to the oe_queue Queue

Add a subscriber to the `oe_queue` queue. This subscriber will perform explicit dequeues of events. A subscriber rule is used to dequeue any events where the `action` value is not `apply`. If the `action` value is `apply` for an event, then the event is ignored by the subscriber. Such events are dequeued and processed by the `apply` process.

```

*/

DECLARE
  subscriber SYS.AQ$_AGENT;
BEGIN
  subscriber := SYS.AQ$_AGENT('explicit_dq', NULL, NULL);
  SYS.DBMS_AQADM.ADD_SUBSCRIBER(
    queue_name => 'strmadmin.oe_queue',
    subscriber => subscriber,
    rule       => 'oe.get_oe_action(tab.user_data) != 'APPLY''');
END;
/

/*

```

Step 5 Create a Procedure to Dequeue Events Explicitly

Connect as `oe`.

```

*/

CONNECT oe/oe@oedb.net

/*

```

Create a PL/SQL procedure called `explicit_dq` to dequeue events explicitly using the subscriber created in Step 4 on page 19-25.

Note:

- This procedure commits after the dequeue of the events. The commit informs the queue that the dequeued messages have been consumed successfully by this subscriber.
 - This procedure can process multiple transactions and uses two exception handlers. The first exception handler `next_trans` moves to the next transaction while the second exception handler `no_messages` exits the loop when there are no more messages.
-
-

*/

```

CREATE OR REPLACE PROCEDURE oe.explicit_dq (consumer IN VARCHAR2) AS
  deqopt      DBMS_AQ.DEQUEUE_OPTIONS_T;
  mprop       DBMS_AQ.MESSAGE_PROPERTIES_T;
  msgid       RAW(16);
  payload     SYS.AnyData;
  new_messages BOOLEAN := TRUE;
  ord         oe.order_event_typ;
  cust        oe.customer_event_typ;
  tc          pls_integer;
  next_trans  EXCEPTION;
  no_messages EXCEPTION;
  pragma exception_init (next_trans, -25235);
  pragma exception_init (no_messages, -25228);
BEGIN
  deqopt.consumer_name := consumer;
  deqopt.wait := 1;
  WHILE (new_messages) LOOP
    BEGIN
      DBMS_AQ.DEQUEUE(
        queue_name      => 'strmadmin.oe_queue',
        dequeue_options => deqopt,
        message_properties => mprop,
        payload         => payload,
        msgid           => msgid);
      COMMIT;
      deqopt.navigation := DBMS_AQ.NEXT;
      DBMS_OUTPUT.PUT_LINE('Event Dequeued');
      DBMS_OUTPUT.PUT_LINE('Type Name := ' || payload.GetTypeName);
    END;
  END LOOP;
END;

```

```

IF (payload.GetTypeName = 'OE.ORDER_EVENT_TYP') THEN
  tc := payload.GetObject(ord);
  DBMS_OUTPUT.PUT_LINE('order_id      - ' || ord.order_id);
  DBMS_OUTPUT.PUT_LINE('order_date   - ' || ord.order_date);
  DBMS_OUTPUT.PUT_LINE('order_mode   - ' || ord.order_mode);
  DBMS_OUTPUT.PUT_LINE('customer_id  - ' || ord.customer_id);
  DBMS_OUTPUT.PUT_LINE('order_status - ' || ord.order_status);
  DBMS_OUTPUT.PUT_LINE('order_total  - ' || ord.order_total);
  DBMS_OUTPUT.PUT_LINE('sales_rep_id - ' || ord.sales_rep_id);
  DBMS_OUTPUT.PUT_LINE('promotion_id - ' || ord.promotion_id);
END IF;
IF (payload.GetTypeName = 'OE.CUSTOMER_EVENT_TYP') THEN
  tc := payload.GetObject(cust);
  DBMS_OUTPUT.PUT_LINE('customer_id      - ' || cust.customer_id);
  DBMS_OUTPUT.PUT_LINE('cust_first_name  - ' || cust.cust_first_name);
  DBMS_OUTPUT.PUT_LINE('cust_last_name   - ' || cust.cust_last_name);
  DBMS_OUTPUT.PUT_LINE('street_address   - ' ||
    cust.cust_address.street_address);
  DBMS_OUTPUT.PUT_LINE('postal_code      - ' ||
    cust.cust_address.postal_code);
  DBMS_OUTPUT.PUT_LINE('city             - ' || cust.cust_address.city);
  DBMS_OUTPUT.PUT_LINE('state_province   - ' ||
    cust.cust_address.state_province);
  DBMS_OUTPUT.PUT_LINE('country_id       - ' ||
    cust.cust_address.country_id);
  DBMS_OUTPUT.PUT_LINE('phone_number1    - ' || cust.phone_numbers(1));
  DBMS_OUTPUT.PUT_LINE('phone_number2    - ' || cust.phone_numbers(2));
  DBMS_OUTPUT.PUT_LINE('phone_number3    - ' || cust.phone_numbers(3));
  DBMS_OUTPUT.PUT_LINE('nls_language     - ' || cust.nls_language);
  DBMS_OUTPUT.PUT_LINE('nls_territory    - ' || cust.nls_territory);
  DBMS_OUTPUT.PUT_LINE('credit_limit      - ' || cust.credit_limit);
  DBMS_OUTPUT.PUT_LINE('cust_email       - ' || cust.cust_email);
  DBMS_OUTPUT.PUT_LINE('account_mgr_id   - ' || cust.account_mgr_id);
END IF;
EXCEPTION
  WHEN next_trans THEN
    deqopt.navigation := DBMS_AQ.NEXT_TRANSACTION;
  WHEN no_messages THEN
    new_messages := FALSE;
    DBMS_OUTPUT.PUT_LINE('No more events');
  END;
END LOOP;
END;
/

```

/*

Step 6 Check Spool Results

Check the `streams_explicit_dq.out` spool file to ensure that all actions completed successfully after this script completes.

*/

```
SET ECHO OFF
SPOOL OFF
```

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

Enqueue Events

Complete the following steps to enqueue non-LCR events and row LCR events into the queue.

1. [Show Output and Spool Results](#)
2. [Enqueue Non-LCR Events to be Dequeued by the Apply Process](#)
3. [Enqueue Non-LCR Events to be Dequeued Explicitly](#)
4. [Enqueue Row LCR Events to be Dequeued by the Apply Process](#)
5. [Check Spool Results](#)

Note:

- It is possible to dequeue user-enqueued LCRs explicitly, but this example does not illustrate this capability.
 - If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the next "END OF SCRIPT" line on page 19-34 into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to the database.
-
-

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


Step 1 Show Output and Spool Results

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

```
*/  
  
SET ECHO ON  
SPOOL streams_enq_deq.out  
  
/*
```

Step 2 Enqueue Non-LCR Events to be Dequeued by the Apply Process

Connect as `oe`.

```
*/  
  
CONNECT oe/oe@oedb.net  
  
/*
```

Enqueue events with `apply` for the `action` value. Based on the apply process rules, the apply process will dequeue and process these events with the `oe.mes_handler` message handler procedure created in "[Create a Message Handler](#)" on page 19-17. The `COMMIT` after the enqueues makes these two enqueues part of the same transaction. An enqueued message is not visible until the session that enqueued it commits the enqueue.

```
*/  
  
BEGIN  
  oe.enq_proc(SYS.AnyData.convertobject(oe.order_event_typ(  
    2500, '05-MAY-01', 'online', 117, 3, 44699, 161, NULL, 'APPLY')));  
END;  
/  
/
```

```

BEGIN
  oe.enq_proc(SYS.AnyData.convertobject(oe.customer_event_typ(
    990,'Hester','Pryne',oe.cust_address_typ('555 Beacon Street','Boston',
    'MA',02109,'US'),oe.phone_list_typ('+1 617 123 4104', '+1 617 083 4381',
    '+1 617 742 5813'),'i','AMERICA',5000,'a@scarlet_letter.com',145,
    NULL,'APPLY')));
END;
/

COMMIT;

/*

```

Step 3 Enqueue Non-LCR Events to be Dequeued Explicitly

Enqueue events with `dequeue` for the `action` value. The `oe.explicit_dq` procedure created in ["Create a Procedure to Dequeue Events Explicitly"](#) on page 19-25 will dequeue these events because the `action` is not `apply`. Based on the `apply` process rules, the `apply` process will ignore these events. The `COMMIT` after the enqueues makes these two enqueues part of the same transaction.

```

*/

BEGIN
  oe.enq_proc(SYS.AnyData.convertobject(oe.order_event_typ(
    2501,'22-JAN-00','direct',117,3,22788,161,NULL,'DEQUEUE')));
END;
/

BEGIN
  oe.enq_proc(SYS.AnyData.convertobject(oe.customer_event_typ(
    991,'Nick','Carraway',oe.cust_address_typ('10th Street',
    11101,'Long Island','NY','US'),oe.phone_list_typ('+1 718 786 2287',
    '+1 718 511 9114', '+1 718 888 4832'),'i','AMERICA',3000,
    'nick@great_gatsby.com',149,NULL,'DEQUEUE')));
END;
/

COMMIT;

/*

```

Step 4 Enqueue Row LCR Events to be Dequeued by the Apply Process

Enqueue row LCR events. The apply process will apply these events directly. Enqueued LCRs should commit at transaction boundaries. In this step, a `COMMIT` statement is run after each enqueue, making each enqueue a separate transaction. However, you can perform multiple LCR enqueues before a commit if there is more than one LCR in a transaction.

Create a row LCR that inserts a row into the `oe.orders` table.

```
*/  
  
DECLARE  
  newunit1 SYS.LCR$_ROW_UNIT;  
  newunit2 SYS.LCR$_ROW_UNIT;  
  newunit3 SYS.LCR$_ROW_UNIT;  
  newunit4 SYS.LCR$_ROW_UNIT;  
  newunit5 SYS.LCR$_ROW_UNIT;  
  newunit6 SYS.LCR$_ROW_UNIT;  
  newunit7 SYS.LCR$_ROW_UNIT;  
  newunit8 SYS.LCR$_ROW_UNIT;  
  newvals  SYS.LCR$_ROW_LIST;  
BEGIN  
  newunit1 := SYS.LCR$_ROW_UNIT(  
    'ORDER_ID',  
    SYS.AnyData.ConvertNumber(2502),  
    DBMS_LCR.NOT_A_LOB,  
    NULL,  
    NULL);  
  newunit2 := SYS.LCR$_ROW_UNIT(  
    'ORDER_DATE',  
    SYS.AnyData.ConvertTimestampLITZ('04-NOV-00'),  
    DBMS_LCR.NOT_A_LOB,  
    NULL,  
    NULL);  
  newunit3 := SYS.LCR$_ROW_UNIT(  
    'ORDER_MODE',  
    SYS.AnyData.ConvertVarchar2('online'),  
    DBMS_LCR.NOT_A_LOB,  
    NULL,  
    NULL);
```

```

newunit4 := SYS.LCR$_ROW_UNIT(
  'CUSTOMER_ID',
  SYS.AnyData.ConvertNumber(145),
  DBMS_LCR.NOT_A_LOB,
  NULL,
  NULL);
newunit5 := SYS.LCR$_ROW_UNIT(
  'ORDER_STATUS',
  SYS.AnyData.ConvertNumber(3),
  DBMS_LCR.NOT_A_LOB,
  NULL,
  NULL);
newunit6 := SYS.LCR$_ROW_UNIT(
  'ORDER_TOTAL',
  SYS.AnyData.ConvertNumber(35199),
  DBMS_LCR.NOT_A_LOB,
  NULL,
  NULL);
newunit7 := SYS.LCR$_ROW_UNIT(
  'SALES_REP_ID',
  SYS.AnyData.ConvertNumber(160),
  DBMS_LCR.NOT_A_LOB,
  NULL,
  NULL);
newunit8 := SYS.LCR$_ROW_UNIT(
  'PROMOTION_ID',
  SYS.AnyData.ConvertNumber(1),
  DBMS_LCR.NOT_A_LOB,
  NULL,
  NULL);
newvals := SYS.LCR$_ROW_LIST(newunit1,newunit2,newunit3,newunit4,
                             newunit5,newunit6,newunit7,newunit8);

oe.enq_row_lcr(
  source_dbname => 'OEDB.NET',
  cmd_type      => 'INSERT',
  obj_owner     => 'OE',
  obj_name      => 'ORDERS',
  old_vals      => NULL,
  new_vals      => newvals);
END;
/
COMMIT;

/*

```

Create a row LCR that updates the row inserted into the `oe.orders` table previously.

```
*/

DECLARE
  oldunit1 SYS.LCR$_ROW_UNIT;
  oldunit2 SYS.LCR$_ROW_UNIT;
  oldvals  SYS.LCR$_ROW_LIST;
  newunit1 SYS.LCR$_ROW_UNIT;
  newvals  SYS.LCR$_ROW_LIST;
BEGIN
  oldunit1 := SYS.LCR$_ROW_UNIT(
    'ORDER_ID',
    SYS.AnyData.ConvertNumber(2502),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  oldunit2 := SYS.LCR$_ROW_UNIT(
    'ORDER_TOTAL',
    SYS.AnyData.ConvertNumber(35199),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  oldvals := SYS.LCR$_ROW_LIST(oldunit1,oldunit2);
  newunit1 := SYS.LCR$_ROW_UNIT(
    'ORDER_TOTAL',
    SYS.AnyData.ConvertNumber(5235),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  newvals := SYS.LCR$_ROW_LIST(newunit1);
  oe.enq_row_lcr(
    source_dbname => 'OEDB.NET',
    cmd_type      => 'UPDATE',
    obj_owner     => 'OE',
    obj_name      => 'ORDERS',
    old_vals      => oldvals,
    new_vals      => newvals);
END;
/
COMMIT;

/*
```

Step 5 Check Spool Results

Check the `streams_enq_deq.out` spool file to ensure that all actions completed successfully after this script completes.

```
*/  
  
SET ECHO OFF  
SPOOL OFF  
  
/***** END OF SCRIPT *****/
```

Dequeue Events Explicitly and Query for Applied Events

Complete the following steps to dequeue the events explicitly and query the events that were applied by the apply process. These events were enqueued in the ["Enqueue Events"](#) on page 19-28.

Step 1 Run the Procedure to Dequeue Events Explicitly

Run the procedure you created in ["Create a Procedure to Dequeue Events Explicitly"](#) on page 19-25 and specify the consumer of the events you want to dequeue. In this case, the consumer is the subscriber you added in ["Add a Subscriber to the oe_queue Queue"](#) on page 19-25. In this example, events that are not dequeued explicitly by this procedure are dequeued by the apply process.

```
CONNECT oe/oe@oedb.net  
  
SET SERVEROUTPUT ON SIZE 100000  
  
EXEC oe.explicit_dq('explicit_dq');
```

You should see the non-LCR events that were enqueued in ["Enqueue Non-LCR Events to be Dequeued Explicitly"](#) on page 19-30.

Step 2 Query for Applied Events

Query the `oe.orders` and `oe.customers` table to see the rows corresponding to the events applied by the apply process:

```
SELECT * FROM oe.orders WHERE order_id = 2500;  
  
SELECT cust_first_name, cust_last_name, cust_email  
FROM oe.customers WHERE customer_id = 990;  
  
SELECT * FROM oe.orders WHERE order_id = 2502;
```

You should see the non-LCR event that was enqueued in "[Enqueue Non-LCR Events to be Dequeued by the Apply Process](#)" on page 19-29 and the row LCR events that were enqueued in "[Enqueue Row LCR Events to be Dequeued by the Apply Process](#)" on page 19-31.

Enqueue and Dequeue Events Using JMS

This example enqueues non-LCR events and row LCR events into the queue using JMS. Then, this example dequeues these events from the queue using JMS.

Complete the following steps:

1. [Run the catxldr.sql Script](#)
2. [Create the Types for User Events](#)
3. [Set the CLASSPATH](#)
4. [Create Java Classes that Map to the Oracle Object Types](#)
5. [Create a Java Code for Enqueuing Messages](#)
6. [Create a Java Code for Dequeuing Messages](#)
7. [Compile the Scripts](#)
8. [Run the Enqueue Program](#)
9. [Run the Dequeue Program](#)

Step 1 Run the catxldr.sql Script

For this example to complete successfully, the LCR schema must be loaded into the SYS schema using the `catxldr.sql` script in Oracle home in the `rdbms/admin/` directory. Run this script now if it has not been run already.

For example, if your Oracle home directory is `/usr/oracle`, then enter the following to run the script:

```
CONNECT SYS/CHANGE_ON_INSTALL AS SYSDBA

@/usr/oracle/rdbms/admin/catxldr.sql
```

Step 2 Create the Types for User Events

```
CONNECT oe/oe
```

```
CREATE TYPE address AS OBJECT (street VARCHAR (30), num NUMBER)
/
```

```
CREATE TYPE person AS OBJECT (name VARCHAR (30), home ADDRESS)
/
```

Step 3 Set the CLASSPATH

The following jar and zip files should be in the CLASSPATH based on the release of JDK you are using.

Also, make sure LD_LIBRARY_PATH (Solaris) or PATH (Windows NT) has \$ORACLE_HOME/lib set.

```
-- For JDK1.3.x
$ORACLE_HOME/jdbc/lib/classes12.zip
$ORACLE_HOME/rdbms/jlib/aqapi13.jar
$ORACLE_HOME/rdbms/jlib/jmscommon.jar
$ORACLE_HOME/rdbms/jlib/xdb.jar
$ORACLE_HOME/xdk/lib/xmlparserv2.jar
$ORACLE_HOME/jlib/jndi.jar
```

```
-- For JDK1.2.x
$ORACLE_HOME/jdbc/lib/classes12.zip
$ORACLE_HOME/rdbms/jlib/aqapi12.jar
$ORACLE_HOME/rdbms/jlib/jmscommon.jar
$ORACLE_HOME/rdbms/jlib/xdb.jar
$ORACLE_HOME/xdk/lib/xmlparserv2.jar
$ORACLE_HOME/jlib/jndi.jar
```

```
-- For JDK1.1.x
$ORACLE_HOME/jdbc/lib/classes111.zip
$ORACLE_HOME/rdbms/jlib/aqapi11.jar
$ORACLE_HOME/rdbms/jlib/jmscommon.jar
$ORACLE_HOME/rdbms/jlib/xdb.jar
$ORACLE_HOME/xdk/lib/xmlparserv2.jar
$ORACLE_HOME/jlib/jndi.jar
```


Step 4 Create Java Classes that Map to the Oracle Object Types

First, create a file `input.typ` with the following lines:

```
SQL PERSON AS JPerson
SQL ADDRESS AS JAddress
```

Then, run `Jpublisher`.

```
jpub -input=input.typ -user=OE/OE
```

Completing these actions generates two Java classes named `JPerson` and `JAddress` for the `person` and `address` types, respectively.

Step 5 Create a Java Code for Enqueuing Messages

This program uses the Oracle JMS API to publish messages into a Streams topic.

This program does the following:

- Publishes a non-LCR based ADT message to the topic
- Publishes a JMS text message to a topic
- Publish an LCR based message to the topic

```
import oracle.AQ.*;
import oracle.jms.*;
import javax.jms.*;
import java.lang.*;
import oracle.xdb.*;

public class StreamsEng
{
    public static void main (String args [])
        throws java.sql.SQLException, ClassNotFoundException, JMSEException
    {
        TopicConnectionFactory tc_fact= null;
        TopicConnection      t_conn = null;
        TopicSession         t_sess = null;

        try
        {
            if (args.length < 3 )
                System.out.println("Usage:java filename [SID] [HOST] [PORT]");
            else
            {
                /* Create the TopicConnectionFactory
```

```

    * Only the JDBC OCI driver can be used to access Streams through JMS
    */
    tc_fact = AQjmsFactory.getTopicConnectionFactory(
        args[1], args[0], Integer.parseInt(args[2]), "oci8");

    t_conn = tc_fact.createTopicConnection( "OE","OE");

    /* Create a Topic Session */
    t_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);

    /* Start the connection */
    t_conn.start() ;

    /* Publish non-LCR based messages */
    publishUserMessages(t_sess);

    /* Publish LCR based messages */
    publishLcrMessages(t_sess);

    t_sess.close() ;
    t_conn.close() ;
    System.out.println("End of StreamsEnq Demo") ;
}
}
catch (Exception ex)
{
    System.out.println("Exception-1: " + ex);
    ex.printStackTrace();
}
}

/*
 * publishUserMessages - this method publishes an ADT message and a
 * JMS text message to a streams topic
 */
public static void publishUserMessages(TopicSession t_sess) throws Exception
{
    Topic          topic      = null;
    TopicPublisher t_pub      = null;
    JPerson        pers       = null;
    JAddress        addr      = null;
    TextMessage    t_msg      = null;
    AdtMessage      adt_msg    = null;
    AQjmsAgent     agent      = null;
    AQjmsAgent[]   recipList = null;

```

```
try
{
    /* Get the topic */
    topic = ((AQjmsSession)t_sess).getTopic("strmadmin", "oe_queue");

    /* Create a publisher */
    t_pub = t_sess.createPublisher(topic);

    /* Agent to access oe_queue */
    agent = new AQjmsAgent("explicit_eng", null);

    /* Create a PERSON adt message */
    adt_msg = ((AQjmsSession)t_sess).createAdtMessage();

    pers = new JPerson();
    addr = new JAddress();

    addr.setNum(new java.math.BigDecimal(500));
    addr.setStreet("Oracle Pkwy");

    pers.setName("Mark");
    pers.setHome(addr);

    /* Set the payload in the message */
    adt_msg.setAdtPayload(pers);

    ((AQjmsMessage)adt_msg).setSenderID(agent);

    System.out.println("Publish message 1 -type PERSON\n");

    /* Create the recipient list */
    recipList = new AQjmsAgent[1];
    recipList[0] = new AQjmsAgent("explicit_dq", null);

    /* Publish the message */
    ((AQjmsTopicPublisher)t_pub).publish(topic, adt_msg, recipList);

    t_sess.commit();

    t_msg = t_sess.createTextMessage();

    t_msg.setText("Test message");
}
```

```

t_msg.setStringProperty("color", "BLUE");
t_msg.setIntProperty("year", 1999);

((AQjmsMessage)t_msg).setSenderID(agent);

System.out.println("Publish message 2 -type JMS TextMessage\n");

/* Publish the message */
((AQjmsTopicPublisher)t_pub).publish(topic, t_msg, recipList);

t_sess.commit();

}
catch (JMSEException jms_ex)
{
    System.out.println("JMS Exception: " + jms_ex);

    if(jms_ex.getLinkedException() != null)
        System.out.println("Linked Exception: " + jms_ex.getLinkedException());
}
}

/*
 * publishLcrMessages - this method publishes an XML LCR message to a
 * streams topic
 */
public static void publishLcrMessages(TopicSession t_sess) throws Exception
{
    Topic            topic      = null;
    TopicPublisher   t_pub      = null;
    XMLType          xml_lcr    = null;
    AdtMessage       adt_msg     = null;
    AQjmsAgent       agent      = null;
    StringBuffer     lcr_data   = null;
    AQjmsAgent[]     recipList  = null;
    java.sql.Connection db_conn  = null;

    try
    {
        /* Get the topic */
        topic = ((AQjmsSession)t_sess).getTopic("strmadmin", "oe_queue");

```

```

/* Create a publisher */
t_pub = t_sess.createPublisher(topic);

/* Get the JDBC connection */
db_conn = ((AQjmsSession)t_sess).getDBCConnection();

/* Agent to access oe_queue */
agent = new AQjmsAgent("explicit_enq", null);

/* Create a adt message */
adt_msg = ((AQjmsSession)t_sess).createAdtMessage();

/* Create the LCR representation in XML */
lcr_data = new StringBuffer();

lcr_data.append("<ROW_LCR ");
lcr_data.append("xmlns='http://xmlns.oracle.com/streams/schemas/lcr' \n");
lcr_data.append("xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance' \n");
lcr_data.append("xsi:schemaLocation='http://xmlns.oracle.com/streams/schemas/lcr
http://xmlns.oracle.com/streams/schemas/lcr/streams_lcr.xsd' ");
lcr_data.append("> \n");

lcr_data.append("<source_database_name>source_dbname</source_database_name> \n");
lcr_data.append("<command_type>INSERT</command_type> \n");
lcr_data.append("<object_owner>Ram</object_owner> \n");
lcr_data.append("<object_name>Emp</object_name> \n");
lcr_data.append("<tag>0ABC</tag> \n");
lcr_data.append("<transaction_id>0.0.0</transaction_id> \n");
lcr_data.append("<scn>0</scn> \n");
lcr_data.append("<old_values> \n");
lcr_data.append("<old_value> \n");
lcr_data.append("<column_name>C01</column_name> \n");
lcr_data.append("<data><varchar2>Clob old</varchar2></data> \n");
lcr_data.append("</old_value> \n");
lcr_data.append("<old_value> \n");
lcr_data.append("<column_name>C02</column_name> \n");
lcr_data.append("<data><varchar2>A123FF</varchar2></data> \n");
lcr_data.append("</old_value> \n");
lcr_data.append("<old_value> \n");
lcr_data.append("<column_name>C03</column_name> \n");
lcr_data.append("<data> \n");
lcr_data.append("<date><value>1997-11-24</value><format>YYYY-MM-DD</format></date> \n");
lcr_data.append("</data> \n");
lcr_data.append("</old_value> \n");

```

```

lcr_data.append("<old_value> \n");
lcr_data.append("<column_name>C04</column_name> \n");
lcr_data.append("<data> \n");

lcr_data.append("<timestamp><value>1999-05-31T13:20:00.000</value><format>SYYYY-MM-DD'T'HH24:MI:
SS.FF</format></timestamp> \n");
lcr_data.append("</data> \n");
lcr_data.append("</old_value> \n");
lcr_data.append("<old_value> \n");
lcr_data.append("<column_name>C05</column_name> \n");
lcr_data.append("<data><raw>ABCDE</raw></data> \n");
lcr_data.append("</old_value> \n");
lcr_data.append("</old_values> \n");
lcr_data.append("<new_values> \n");
lcr_data.append("<new_value> \n");
lcr_data.append("<column_name>C01</column_name> \n");
lcr_data.append("<data><varchar2>A123FF</varchar2></data> \n");
lcr_data.append("</new_value> \n");
lcr_data.append("<new_value> \n");
lcr_data.append("<column_name>C02</column_name> \n");
lcr_data.append("<data><number>35.23</number></data> \n");
lcr_data.append("</new_value> \n");
lcr_data.append("<new_value> \n");
lcr_data.append("<column_name>C03</column_name> \n");
lcr_data.append("<data><number>-100000</number></data> \n");
lcr_data.append("</new_value> \n");
lcr_data.append("<new_value> \n");
lcr_data.append("<column_name>C04</column_name> \n");
lcr_data.append("<data><varchar>Hel lo</varchar></data> \n");
lcr_data.append("</new_value> \n");
lcr_data.append("<new_value> \n");
lcr_data.append("<column_name>C05</column_name> \n");
lcr_data.append("<data><char>wor ld</char></data> \n");
lcr_data.append("</new_value> \n");
lcr_data.append("</new_values> \n");
lcr_data.append("</ROW_LCR>");

/* Create the XMLType containing the LCR */
xml_lcr = oracle.xdb.XMLType.createXML(db_conn, lcr_data.toString());

/* Set the payload in the message */
adt_msg.setAdtPayload(xml_lcr);

((AQjmsMessage)adt_msg).setSenderID(agent);

```

```

System.out.println("Publish message 3 - XMLType containing LCR ROW\n");

/* Create the recipient list */
recipList = new AQjmsAgent[1];
recipList[0] = new AQjmsAgent("explicit_dq", null);

/* Publish the message */
((AQjmsTopicPublisher)t_pub).publish(topic, adt_msg, recipList);

t_sess.commit();

}
catch (JMSEException jms_ex)
{
    System.out.println("JMS Exception: " + jms_ex);

    if(jms_ex.getLinkedException() != null)
        System.out.println("Linked Exception: " + jms_ex.getLinkedException());
}
}
}

```

Step 6 Create a Java Code for Dequeuing Messages

This program uses Oracle JMS API to receive messages from a Streams topic.

This program does the following:

- Registers mappings for `person`, `address` and `XMLType` in JMS typemap
- Receives LCR messages from a streams topic
- Receives user ADT messages from a streams topic

```

import oracle.AQ.*;
import oracle.jms.*;
import javax.jms.*;
import java.lang.*;
import oracle.xdb.*;
import java.sql.SQLException;

```

```

public class StreamsDeq
{
    public static void main (String args [])
        throws java.sql.SQLException, ClassNotFoundException, JMSEException
    {
        TopicConnectionFactory tc_fact= null;
        TopicConnection      t_conn = null;
        TopicSession         t_sess = null;

        try
        {
            if (args.length < 3 )
                System.out.println("Usage:java filename [SID] [HOST] [PORT]");
            else
            {
                /* Create the TopicConnectionFactory
                 * Only the JDBC OCI driver can be used to access Streams through JMS
                 */
                tc_fact = AQjmsFactory.getTopicConnectionFactory(
                    args[1], args[0], Integer.parseInt(args[2]), "oci8");

                t_conn = tc_fact.createTopicConnection( "OE","OE");

                /* Create a Topic Session */
                t_sess = t_conn.createTopicSession(true, Session.CLIENT_ACKNOWLEDGE);

                /* Start the connection */
                t_conn.start() ;

                receiveMessages(t_sess);

                t_sess.close() ;
                t_conn.close() ;
                System.out.println("\nEnd of StreamsDeq Demo") ;
            }
        }
        catch (Exception ex)
        {
            System.out.println("Exception-1: " + ex);
            ex.printStackTrace();
        }
    }
}

```



```
/*
 * receiveMessages -This method receives messages from the Streams queue
 */
public static void receiveMessages(TopicSession t_sess) throws Exception
{
    Topic          topic    = null;
    JPerson        pers     = null;
    JAddress       addr     = null;
    XMLType        xtype    = null;
    TextMessage    t_msg    = null;
    AdtMessage     adt_msg  = null;
    Message        jms_msg  = null;
    TopicReceiver  t_recv   = null;
    int            i        = 0;
    java.util.Dictionary map= null;

    try
    {
        /* Get the topic */
        topic = ((AQjmsSession)t_sess).getTopic("strmadmin", "oe_queue");

        /* Create a TopicReceiver to receive messages for consumer "jms_recv */
        t_recv = ((AQjmsSession)t_sess).createTopicReceiver(topic,
                                                            "jms_recv", null);

        map = ((AQjmsSession)t_sess).getTypeMap();

        /* Register mappings for ADDRESS and PERSON in the JMS typemap */
        map.put("OE.PERSON", Class.forName("JPerson"));
        map.put("OE.ADDRESS", Class.forName("JAddress"));

        /* Register mapping for XMLType in the TypeMap - required for LCRs */
        map.put("SYS.XMLTYPE", Class.forName("oracle.xdb.XMLTypeFactory"));

        System.out.println("Receive messages ... \n");

        do
        {
            try
            {
                {
                    jms_msg = (t_recv.receive(10));

                    i++;
                }
            }
        }
    }
}
```

```
        /* Set navigation mode to NEXT_MESSAGE */

((AQjmsTopicReceiver)t_recv).setNavigationMode(AQjmsConstants.NAVIGATION_NEXT_MESSAGE);
    }
    catch (JMSEException jms_ex2)
    {
        if((jms_ex2.getLinkedException() != null) &&
            (jms_ex2.getLinkedException() instanceof SQLException))
        {
            SQLException sql_ex2 =(SQLException)(jms_ex2.getLinkedException());

            /* End of current transaction group
             * Use NEXT_TRANSACTION navigation mode
             */
            if(sql_ex2.getErrorCode() == 25235)
            {

((AQjmsTopicReceiver)t_recv).setNavigationMode(AQjmsConstants.NAVIGATION_NEXT_TRANSACTION);

                continue;
            }
            else
                throw jms_ex2;
        }
        else
            throw jms_ex2;
    }

    if(jms_msg == null)
    {
        System.out.println("\nNo more messages");
    }
    else
    {
        if(jms_msg instanceof AdtMessage)
        {
            adt_msg = (AdtMessage)jms_msg;

            System.out.println("Retrieved message " + i + ": " +
                adt_msg.getAdtPayload());
        }
    }
}
```

```
if(adMsg.getAdtPayload() instanceof JPerson)
{
    pers =(JPerson)( adMsg.getAdtPayload());

    System.out.println("PERSON: Name: " + pers.getName());
}
else if(adMsg.getAdtPayload() instanceof JAddress)
{
    addr =(JAddress)( adMsg.getAdtPayload());

    System.out.println("ADDRESS: Street" + addr.getStreet());
}
else if(adMsg.getAdtPayload() instanceof oracle.xdb.XMLType)
{
    xtype = (XMLType)adMsg.getAdtPayload();

    System.out.println("XMLType: Data: \n" + xtype.getStringVal());
}
System.out.println("Msg id: " + adMsg.getJMSMessageID());
System.out.println();
}
else if(jmsMsg instanceof TextMessage)
{
    tMsg = (TextMessage)jmsMsg;

    System.out.println("Retrieved message " + i + ": " +
        tMsg.getText());

    System.out.println("Msg id: " + tMsg.getJMSMessageID());
    System.out.println();
}
else
    System.out.println("Invalid message type");
}
} while (jmsMsg != null);

t_sess.commit();
}
catch (JMSEException jms_ex)
{
    System.out.println("JMS Exception: " + jms_ex);
}
```

```
        if(jms_ex.getLinkedException() != null)
            System.out.println("Linked Exception: " + jms_ex.getLinkedException());

        t_sess.rollback();
    }
    catch (java.sql.SQLException sql_ex)
    {
        System.out.println("SQL Exception: " + sql_ex);
        sql_ex.printStackTrace();

        t_sess.rollback();
    }
}
```

Step 7 Compile the Scripts

```
javac StreamsEnq.java StreamsDeq.java JPerson.java JAddress.java
```

Step 8 Run the Enqueue Program

```
java StreamsEnq ORACLE_SID HOST PORT
```

For example, if your Oracle SID is `orcl82`, your host is `hq_server`, and your port is 1521, then enter the following:

```
java StreamsEnq orcl82 hq_server 1521
```

Step 9 Run the Dequeue Program

```
java StreamsDeq ORACLE_SID HOST PORT
```

For example, if your Oracle SID is `orcl82`, your host is `hq_server`, and your port is 1520, then enter the following:

```
java StreamsDeq orcl82 hq_server 1521
```

Single Database Capture and Apply Example

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.

This chapter contains these topics:

- [Overview of the Single Database Capture and Apply Example](#)
- [Prerequisites](#)
- [Set Up the Environment](#)
- [Configure Capture and Apply](#)
- [Make DML Changes, Query for Results, and Dequeue Events](#)

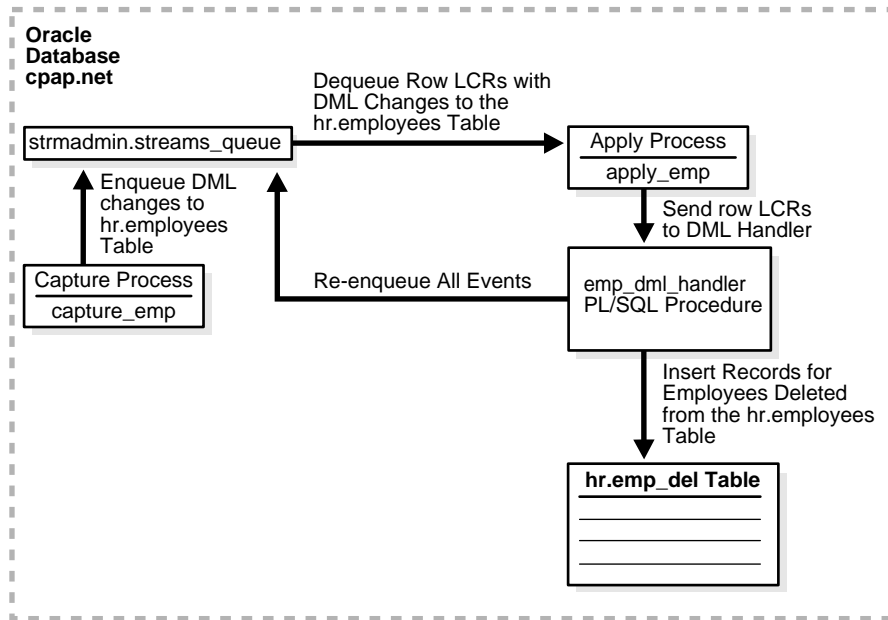
Overview of the Single Database Capture and Apply Example

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`. Then, an apply process dequeues these row LCRs from the same queue and sends them to a DML handler. The DML handler performs the following actions on the captured row LCRs:

- Re-enqueues all captured row LCRs back into the queue. When the row LCRs are captured, they reside in the buffer queue and cannot be dequeued explicitly. After the row LCRs are re-enqueued during by the DML handler, they are available for explicit dequeue by an application. This example does not create the application that dequeues these row LCRs.
- 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.

Figure 20–1 provides an overview of the environment.

Figure 20–1 Single Database Capture and Apply Example



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 the database:
 - `AQ_TM_PROCESSES`: This parameter establishes queue monitor processes. Values from 1 to 10 specify the number of queue monitor processes created to monitor the messages. If `AQ_TM_PROCESSES` is not specified or is set to 0, then the queue monitor processes are not created. In this example, `AQ_TM_PROCESSES` should be set to at least 1.

Setting the parameter to 1 or more starts the specified number of queue monitor processes. These queue monitor processes are responsible for managing time-based operations of messages such as delay and expiration, cleaning up retained messages after the specified retention time, and cleaning up consumed messages if the retention time is 0.

- `COMPATIBLE`: This parameter must be set to 9.2.0 or higher.
- `LOG_PARALLELISM`: This parameter must be set to 1 because the database will capture events.

See Also: ["Setting Initialization Parameters Relevant to Streams"](#) on page 11-4 for information about other initialization parameters that are important in a Streams environment

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

See Also: *Oracle9i 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.

Set Up the Environment

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

1. [Show Output and Spool Results](#)
2. [Create the hr.emp_del Table](#)
3. [Set Up Users at cpap.net](#)
4. [Create the Streams Queue at cpap.net](#)
5. [Check the Spool Results](#)

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the next "END OF SCRIPT" line on page 20-9 into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to all of the databases in the environment.

```

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

```

Step 1 Show Output and Spool Results

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

```

*/

SET ECHO ON
SPOOL streams_setup_catapp.out

/*

```

Step 2 Create the hr.emp_del Table

Connect to cpap.net as the hr user.

```
*/
```

```
CONNECT hr/hr@cpap.net
```

```
/*
```

Create the hr.emp_del table. The shape of the emp_del table is the same as the employees table, except for one added timestamp column that will record the date when a row is inserted into the emp_del table.

```
*/
```

```
CREATE TABLE emp_del(  
    employee_id    NUMBER(6),  
    first_name     VARCHAR2(20),  
    last_name      VARCHAR2(25),  
    email          VARCHAR2(25),  
    phone_number   VARCHAR2(20),  
    hire_date      DATE,  
    job_id         VARCHAR2(10),  
    salary         NUMBER(8,2),  
    commission_pct NUMBER(2,2),  
    manager_id     NUMBER(6),  
    department_id  NUMBER(4),  
    timestamp      DATE);
```

```
CREATE UNIQUE INDEX emp_del_id_pk ON emp_del (employee_id);
```

```
ALTER TABLE emp_del ADD (CONSTRAINT emp_del_id_pk PRIMARY KEY (employee_id));
```

```
/*
```

Step 3 Set Up Users at cpap.net

Connect to `cpap.net` as `SYS` user.

```
*/
```

```
CONNECT SYS/CHANGE_ON_INSTALL@cpap.net AS SYSDBA
```

```
/*
```

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

In this example, the Streams administrator will be the apply user for the apply process and must be able to apply changes to the `hr.emp_del` table. Therefore, the Streams administrator is granted `ALL` privileges on this table.

Note:

- To ensure security, use a password other than `strmadminpw` for the Streams administrator.
 - The `SELECT_CATALOG_ROLE` is not required for the Streams administrator. It is granted in this example so that the Streams administrator can monitor the environment easily.
 - If you plan to use the Streams tool in Oracle Enterprise Manager, then grant the Streams administrator `SELECT ANY DICTIONARY` privilege, in addition to the privileges shown in this step.
 - The `ACCEPT` command must appear on a single line in the script.
-
-

See Also: "[Configuring a Streams Administrator](#)" on page 11-2

```
*/

GRANT CONNECT, RESOURCE, SELECT_CATALOG_ROLE
  TO strmadmin IDENTIFIED BY strmadminpw;

ACCEPT streams_tbs PROMPT 'Enter Streams administrator tablespace on cpap.net: '

ALTER USER strmadmin DEFAULT TABLESPACE &streams_tbs
  QUOTA UNLIMITED ON &streams_tbs;

GRANT ALL ON hr.emp_del TO strmadmin;

GRANT EXECUTE ON DBMS_APPLY_ADM          TO strmadmin;
GRANT EXECUTE ON DBMS_AQ                 TO strmadmin;
GRANT EXECUTE ON DBMS_AQADM              TO strmadmin;
GRANT EXECUTE ON DBMS_CAPTURE_ADM        TO strmadmin;
GRANT EXECUTE ON DBMS_FLASHBACK          TO strmadmin;
GRANT EXECUTE ON DBMS_STREAMS_ADM        TO strmadmin;

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

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

/*
```

Step 4 Create the Streams Queue at cpap.net

Connect to `cpap.net` as the `strmadmin` user.

```
*/
CONNECT strmadmin/strmadminpw@cpap.net
/*
```

Run the `SET_UP_QUEUE` procedure to create a queue named `streams_queue` at `cpap.net`. This queue will function as the Streams queue by holding the captured changes that will be dequeued by an apply process.

Running the `SET_UP_QUEUE` procedure performs the following actions:

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

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

Step 5 Check the Spool Results

Check the `streams_setup_catapp.out` spool file to ensure that all actions finished successfully after this script is completed.

```
*/
SET ECHO OFF
SPOOL OFF

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

Configure Capture and Apply

Complete the following steps to capture changes to the `hr.employees` table and apply these changes on single database in a customized way using a DML handler.

1. [Show Output and Spool Results](#)
2. [Create an Alternate Tablespace for the LogMiner Tables at cpap.net](#)
3. [Specify Supplemental Logging at cpap.net](#)
4. [Configure the Capture Process at cpap.net](#)
5. [Set the Instantiation SCN for the hr.employees Table](#)
6. [Create an Agent Named emp_agent](#)
7. [Create a Queue Subscriber](#)
8. [Create a Procedure to Enqueue Row LCRs](#)
9. [Create the DML Handler Procedure](#)
10. [Set the DML Handler for the hr.employees Table](#)
11. [Create a Procedure to Dequeue the Re-enqueued Events](#)
12. [Configure the Apply Process at cpap.net](#)
13. [Start the Apply Process at cpap.net](#)
14. [Start the Capture Process at cpap.net](#)
15. [Check the Spool Results](#)

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the next "END OF SCRIPT" line on page 20-21 into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to all of the databases in the environment.

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

Step 1 Show Output and Spool Results

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

```
*/  
  
SET ECHO ON  
SPOOL streams_config_capapp.out  
  
/*
```

Step 2 Create an Alternate Tablespace for the LogMiner Tables at cpap.net

By default, the LogMiner tables are in the `SYSTEM` tablespace, but the `SYSTEM` tablespace may not have enough space for these tables once a capture process starts to capture changes. Therefore, you must create an alternate tablespace for the LogMiner tables.

See Also: ["Alternate Tablespace for LogMiner Tables"](#) on page 2-20

Connect to `cpap.net` as `SYS` user.

```
*/  
  
CONNECT SYS/CHANGE_ON_INSTALL@cpap.net AS SYSDBA  
  
/*
```

Create an alternate tablespace for the LogMiner tables.

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

```
*/

ACCEPT tspace_name DEFAULT 'logmrts' PROMPT 'Enter tablespace name (for
example, logmrts): '

ACCEPT db_file_directory DEFAULT '' PROMPT 'Enter the complete path to the
datafile directory (for example, /usr/oracle/dbs): '

ACCEPT db_file_name DEFAULT 'logmrts.dbf' PROMPT 'Enter the name of the
datafile (for example, logmrts.dbf): '

CREATE TABLESPACE &tspace_name DATAFILE '&db_file_directory/&db_file_name'
  SIZE 25 M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;

EXECUTE DBMS_LOGMNR_D.SET_TABLESPACE('&tspace_name');

/*
```

Step 3 Specify Supplemental Logging at cpap.net

Supplemental logging places additional information in the redo log for changes made to tables. The apply process needs this extra information to perform certain operations, such as unique row identification.

The following statement specifies an unconditional supplemental log group for the primary key column in the `hr.employees` table.

See Also:

- ["Supplemental Logging in a Streams Environment"](#) on page 2-11
- ["Specifying Supplemental Logging at a Source Database"](#) on page 12-9


```
*/  
  
ALTER TABLE hr.employees ADD SUPPLEMENTAL LOG GROUP log_group_employees_pk  
  (employee_id) ALWAYS;
```

```
/*
```

Step 4 Configure the Capture Process at cpap.net

Connect to cpap.net as the strmadmin user.

```
*/
```

```
CONNECT strmadmin/strmadminpw@cpap.net
```

```
/*
```

Configure the capture process to capture DML changes to the hr.employees table at cpap.net. This step specifies that DML changes to this table are captured by the capture process and enqueued into the specified queue.

```
*/
```

```
BEGIN  
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(  
    table_name      => 'hr.employees',  
    streams_type    => 'capture',  
    streams_name    => 'capture_emp',  
    queue_name      => 'strmadmin.streams_queue',  
    include_dml     => true,  
    include_ddl     => false);
```

```
END;
```

```
/
```

```
/*
```

Step 5 Set the Instantiation SCN for the hr.employees Table

Because this example captures and applies changes in a single database, no instantiation is necessary. However, the apply process at the `cpap.net` database still must be instructed to apply changes that were made to the `hr.employees` table after a certain system change number (SCN).

This example uses the `GET_SYSTEM_CHANGE_NUMBER` function in the `DBMS_FLASHBACK` package to obtain the current SCN for the database. This SCN is used to run the `SET_TABLE_INSTANTIATION_SCN` procedure in the `DBMS_APPLY_ADM` package.

The `SET_TABLE_INSTANTIATION_SCN` procedure controls which LCRs for a table are ignored by an apply process and which LCRs for a table are applied by an apply process. If the commit SCN of an LCR for a table from a source database is less than or equal to the instantiation SCN for that table at a destination database, then the apply process at the destination database discards the LCR. Otherwise, the apply process applies the LCR. In this example, the `cpap.net` database is both the source database and the destination database.

The apply process will apply transactions to the `hr.employees` table with SCNs that were committed after SCN obtained in this step.

Note: The `hr.employees` table must also be prepared for instantiation. This preparation was done automatically when the capture process was configured with a rule to capture DML changes to the `hr.employees` table in Step 4.

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

Step 6 Create an Agent Named emp_agent

This example uses an agent named `emp_agent` for explicit enqueue into and dequeue from the `streams_queue`. Because the `strmadmin` user owns the queue table for this queue, the `strmadmin` user is a secure user of the queue. This step creates the agent named `emp_agent` and associates this agent with the `strmadmin` user, which allows the agent to be used for enqueues into and dequeues from the secure queue.

```
*/

BEGIN
  DBMS_AQADM.CREATE_AQ_AGENT(
    agent_name => 'emp_agent');
  DBMS_AQADM.ENABLE_DB_ACCESS(
    agent_name => 'emp_agent',
    db_username => 'strmadmin');
END;
/

/*
```

Step 7 Create a Queue Subscriber

Create a subscriber that can be used by an application to dequeue the re-enqueued events. At least one subscriber must be specified before the events can be re-enqueued into the queue.

```
*/

DECLARE
  subscriber SYS.AQ$_AGENT;
BEGIN
  subscriber := SYS.AQ$_AGENT('emp_agent', NULL, NULL);
  SYS.DBMS_AQADM.ADD_SUBSCRIBER(
    queue_name      => 'strmadmin.streams_queue',
    subscriber      => subscriber,
    rule            => NULL,
    transformation  => NULL);
END;
/

/*
```

Step 8 Create a Procedure to Enqueue Row LCRs

This step creates the `enq_row_lcr` procedure. This procedure will be used in the DML handler procedure created in Step 9 to enqueue row LCRs that contain changes to the `hr.employees` table.

```

*/

CREATE OR REPLACE PROCEDURE enq_row_lcr(in_any IN SYS.ANYDATA) IS
  enqopt      DBMS_AQ.ENQUEUE_OPTIONS_T;
  mprop      DBMS_AQ.MESSAGE_PROPERTIES_T;
  recipients  DBMS_AQ.AQ$_RECIPIENT_LIST_T;
  enq_eventid RAW(16);
BEGIN
  mprop.SENDER_ID := SYS.AQ$_AGENT(
    name      => 'emp_agent',
    address   => NULL,
    protocol  => NULL);
  recipients(1) := SYS.AQ$_AGENT(
    name      => 'emp_agent',
    address   => NULL,
    protocol  => NULL);
  mprop.RECIPIENT_LIST := recipients;
  DBMS_AQ.ENQUEUE(
    queue_name      => 'strmadmin.streams_queue',
    enqueue_options => enqopt,
    message_properties => mprop,
    payload         => in_any,
    msgid          => enq_eventid);
END;
/

/*

```

Step 9 Create the DML Handler Procedure

This step creates the `emp_dml_handler` procedure. This procedure will be the DML handler for DML changes to the `hr.employees` table. It performs the following actions:

- Re-enqueues all row LCRs back into `streams_queue` using the `enq_row_lcr` procedure created in Step 8.
- Converts any row LCR containing a `DELETE` command type into an `INSERT` row LCR and then inserts the converted row LCR into the `hr.emp_del` table by executing the row LCR.

```

*/

CREATE OR REPLACE PROCEDURE emp_dml_handler(in_any IN SYS.ANYDATA) IS
  lcr          SYS.LCR$_ROW_RECORD;
  rc           PLS_INTEGER;
  command      VARCHAR2(10);
  old_values   SYS.LCR$_ROW_LIST;
BEGIN
  -- Re-enqueue the row LCR for explicit dequeue by another application
  enq_row_lcr(in_any);
  -- Access the LCR
  rc := in_any.GETOBJECT(lcr);
  -- Get the object command type
  command := lcr.GET_COMMAND_TYPE();
  -- Check for DELETE command on the hr.employees table
  IF command = 'DELETE' THEN
    -- Set the command_type in the row LCR to INSERT
    lcr.SET_COMMAND_TYPE('INSERT');
    -- Set the object_name in the row LCR to EMP_DEL
    lcr.SET_OBJECT_NAME('EMP_DEL');
    -- Get the old values in the row LCR
    old_values := lcr.GET_VALUES('old');
    -- Set the old values in the row LCR to the new values in the row LCR
    lcr.SET_VALUES('new', old_values);
    -- Set the old values in the row LCR to NULL
    lcr.SET_VALUES('old', NULL);
    -- Add a SYSDATE value for the timestamp column
    lcr.ADD_COLUMN('new', 'TIMESTAMP', SYS.AnyData.ConvertDate(SYSDATE));
    -- Apply the row LCR as an INSERT into the EMP_DEL table
    lcr.EXECUTE(true);
  END IF;
END;
/

/*

```

Step 10 Set the DML Handler for the hr.employees Table

Set the DML handler for the `hr.employees` table to the procedure created in Step 9. Notice that the DML handler must be set separately for each possible operation on the table: INSERT, UPDATE, and DELETE.

```

*/

BEGIN
  DBMS_APPLY_ADM.SET_DML_HANDLER(
    object_name      => 'hr.employees',
    object_type      => 'TABLE',
    operation_name   => 'INSERT',
    error_handler    => false,
    user_procedure   => 'strmadmin.emp_dml_handler',
    apply_database_link => NULL);
END;
/

BEGIN
  DBMS_APPLY_ADM.SET_DML_HANDLER(
    object_name      => 'hr.employees',
    object_type      => 'TABLE',
    operation_name   => 'UPDATE',
    error_handler    => false,
    user_procedure   => 'strmadmin.emp_dml_handler',
    apply_database_link => NULL);
END;
/

BEGIN
  DBMS_APPLY_ADM.SET_DML_HANDLER(
    object_name      => 'hr.employees',
    object_type      => 'TABLE',
    operation_name   => 'DELETE',
    error_handler    => false,
    user_procedure   => 'strmadmin.emp_dml_handler',
    apply_database_link => NULL);
END;
/

/*

```

Step 11 Create a Procedure to Dequeue the Re-enqueued Events

The `emp_dq` procedure created in this step can be used to dequeue the events that are re-enqueued by the DML handler created in Step 9. When the `emp_dq` procedure is executed, it dequeues each row LCR in the queue and displays the type of command in the row LCR, either `INSERT`, `UPDATE`, or `DELETE`. Any information in the row LCRs can be accessed and displayed, not just the command type.

See Also: ["Displaying Detailed Information About Apply Errors"](#) on page 17-37 for more information about displaying information in LCRs

```

*/

CREATE OR REPLACE PROCEDURE emp_dq (consumer IN VARCHAR2) AS
  deqopt      DBMS_AQ.DEQUEUE_OPTIONS_T;
  mprop       DBMS_AQ.MESSAGE_PROPERTIES_T;
  msgid       RAW(16);
  payload     SYS.AnyData;
  new_messages BOOLEAN := TRUE;
  row_lcr     SYS.LCR$_ROW_RECORD;
  tc          pls_integer;
  next_trans  EXCEPTION;
  no_messages EXCEPTION;
  pragma exception_init (next_trans, -25235);
  pragma exception_init (no_messages, -25228);
BEGIN
  deqopt.consumer_name := consumer;
  deqopt.wait := 1;
  WHILE (new_messages) LOOP
    BEGIN
      DBMS_AQ.DEQUEUE(
        queue_name      => 'strmadmin.streams_queue',
        dequeue_options => deqopt,
        message_properties => mprop,
        payload         => payload,
        msgid           => msgid);
      COMMIT;
      deqopt.navigation := DBMS_AQ.NEXT;
      IF (payload.GetTypeName = 'SYS.LCR$_ROW_RECORD') THEN
        tc := payload.GetObject(row_lcr);
        DBMS_OUTPUT.PUT_LINE(row_lcr.GET_COMMAND_TYPE || ' row LCR dequeued');
      END IF;
      EXCEPTION
        WHEN next_trans THEN
          deqopt.navigation := DBMS_AQ.NEXT_TRANSACTION;
        WHEN no_messages THEN
          new_messages := FALSE;
          DBMS_OUTPUT.PUT_LINE('No more events');
    END;
  END LOOP;
END;

```

/

/*

Step 12 Configure the Apply Process at cpap.net

Configure an apply process to apply DML changes to the `hr.employees` table. Although the DML handler for the apply process causes deleted employees to be inserted into the `emp_del` table, this rule specifies the `employees` table, because the row LCRs in the queue contain changes to the `employees` table, not the `emp_del` table.

*/

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name      => 'hr.employees',
    streams_type    => 'apply',
    streams_name    => 'apply_emp',
    queue_name      => 'strmadmin.streams_queue',
    include_dml     => true,
    include_ddl     => false,
    source_database => 'cpap.net');
```

END;

/

/*

Step 13 Start the Apply Process at cpap.net

Set the `disable_on_error` parameter to `n` so that the apply process will not be disabled if it encounters an error, and start the apply process at `cpap.net`.

*/

```
BEGIN
  DBMS_APPLY_ADM.SET_PARAMETER(
    apply_name      => 'apply_emp',
    parameter       => 'disable_on_error',
    value           => 'n');
```

END;

/


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

/*
```

Step 14 Start the Capture Process at cpap.net

Start the capture process at cpap.net.

```
*/

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

/*
```

Step 15 Check the Spool Results

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

```
*/

SET ECHO OFF
SPOOL OFF

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

Make DML Changes, Query for Results, and Dequeue Events

Complete the following steps to make DML changes to the `hr.employees` table, query for the resulting inserts into the `hr.emp_del` table and the re-enqueued events in the `streams_queue_table`, and dequeue the events that were re-enqueued by the DML handler.

Step 1 Perform an INSERT, UPDATE, and DELETE on hr.employees

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

```
CONNECT hr/hr@cpap.net
```

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

```
UPDATE hr.employees SET salary=5999 WHERE employee_id=206;  
COMMIT;
```

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

Step 2 Query the hr.emp_del Table and the streams_queue_table

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

```
CONNECT strmadmin/strmadminpw@cpap.net
```

```
SELECT * FROM hr.emp_del;
```

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

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

Step 3 Dequeue Events Re-enqueued by the DML Handler

Use the `emp_dq` procedure to dequeue the events that were re-enqueued by the DML handler.

```
SET SERVEROUTPUT ON SIZE 100000
```

```
EXEC emp_dq('emp_agent');
```

For each row changed by a DML statement, one line is returned, and each line states the command type of the change (either `INSERT`, `UPDATE`, or `DELETE`). If you repeat the query on the queue table in Step 2 after the events are dequeued, then the dequeued events should have been consumed. That is, the `MSG_STATE` should be `PROCESSED` for these events.

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

Simple Single Source Replication Example

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

This chapter contains these topics:

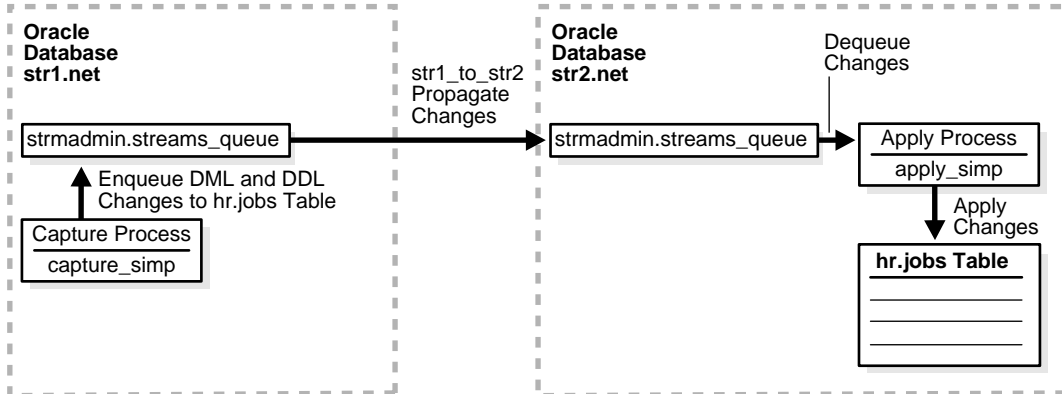
- [Overview of the Simple Single Source Replication Example](#)
- [Prerequisites](#)
- [Set Up Users and Create Queues and Database Links](#)
- [Configure Capture, Propagation, and Apply for Changes to One Table](#)
- [Make Changes to the hr.jobs Table and View Results](#)

Overview of the Simple Single Source Replication Example

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

Figure 21-1 provides an overview of the environment.

Figure 21-1 Simple Example That Shares Data From a Single Source Database



Prerequisites

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

- Set the following initialization parameters to the values indicated:
 - `AQ_TM_PROCESSES`: This parameter establishes queue monitor processes. Values from 1 to 10 specify the number of queue monitor processes created to monitor the messages. If `AQ_TM_PROCESSES` is not specified or is set to 0, then the queue monitor processes are not created. In this example, `AQ_TM_PROCESSES` should be set to at least 1 at each database.

Setting the parameter to 1 or more starts the specified number of queue monitor processes. These queue monitor processes are responsible for managing time-based operations of messages such as delay and expiration, cleaning up retained messages after the specified retention time, and cleaning up consumed messages if the retention time is 0.
 - `GLOBAL_NAMES`: This parameter must be set to `true` at each database that is participating in your Streams environment.
 - `JOB_QUEUE_PROCESSES`: This parameter must be set to at least 2 at each database that is propagating events in your Streams environment. It should be set to the same value as the maximum number of jobs that can run simultaneously plus one. In this example, `str1.net` propagates events. So, `JOB_QUEUE_PROCESSES` must be set to at least 2 at `str1.net`.
 - `COMPATIBLE`: This parameter must be set to 9.2.0 or higher at each database that is participating in your Streams environment.
 - `LOG_PARALLELISM`: This parameter must be set to 1 at each database that captures events. In this example, this parameter must be set to 1 at `str1.net`.

See Also: ["Setting Initialization Parameters Relevant to Streams"](#) on page 11-4 for information about other initialization parameters that are important in a Streams environment

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

See Also: *Oracle9i Database Administrator's Guide* for information about running a database in ARCHIVELOG mode

- Configure your network and Oracle Net so that the `str1.net` database can communicate with the `str2.net` database.
See Also: *Oracle9i Net Services Administrator's Guide*
- This example creates a new user to function as the Streams administrator (`strmadmin`) at each database and prompts you for the tablespace you want to use for this user's data. Before you start this example, either create a new tablespace or identify an existing tablespace for the Streams administrator to use at each database. The Streams administrator should not use the `SYSTEM` tablespace.

Set Up Users and Create Queues and Database Links

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

1. [Show Output and Spool Results](#)
2. [Set Up Users at str1.net](#)
3. [Create the Streams Queue at str1.net](#)
4. [Create the Database Link at str1.net](#)
5. [Set Up Users at str2.net](#)
6. [Set Up the Streams Queue at str2.net](#)
7. [Check the Spool Results](#)

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the next "END OF SCRIPT" line on page 21-10 into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to all of the databases in the environment.

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


Step 1 Show Output and Spool Results

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

```
*/  
  
SET ECHO ON  
SPOOL streams_setup_simple.out  
  
/*
```

Step 2 Set Up Users at str1.net

Connect to `str1.net` as `SYS` user.

```
*/  
  
CONNECT SYS/CHANGE_ON_INSTALL@str1.net AS SYSDBA  
  
/*
```

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

Note:

- To ensure security, use a password other than `strmadminpw` for the Streams administrator.
 - The `SELECT_CATALOG_ROLE` is not required for the Streams administrator. It is granted in this example so that the Streams administrator can monitor the environment easily.
 - If you plan to use the Streams tool in Oracle Enterprise Manager, then grant the Streams administrator `SELECT ANY DICTIONARY` privilege, in addition to the privileges shown in this step.
 - The `ACCEPT` command must appear on a single line in the script.
-
-

See Also: ["Configuring a Streams Administrator" on page 11-2](#)

```
*/

GRANT CONNECT, RESOURCE, SELECT_CATALOG_ROLE
  TO strmadmin IDENTIFIED BY strmadminpw;

ACCEPT streams_tbs PROMPT 'Enter Streams administrator tablespace on str1.net: '

ALTER USER strmadmin DEFAULT TABLESPACE &streams_tbs
  QUOTA UNLIMITED ON &streams_tbs;

GRANT EXECUTE ON DBMS_AQADM          TO strmadmin;
GRANT EXECUTE ON DBMS_CAPTURE_ADM   TO strmadmin;
GRANT EXECUTE ON DBMS_PROPAGATION_ADM TO strmadmin;
GRANT EXECUTE ON DBMS_STREAMS_ADM   TO strmadmin;

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

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

/*
```

Step 3 Create the Streams Queue at str1.net

Connect as the Streams administrator at the database where you want to capture changes. In this example, that database is str1.net.

```
*/

CONNECT strmadmin/strmadminpw@str1.net

/*
```

Run the `SET_UP_QUEUE` procedure to create a queue named `streams_queue` at `str1.net`. This queue will function as the Streams queue by holding the captured changes that will be propagated to other databases.

Running the `SET_UP_QUEUE` procedure performs the following actions:

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

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

Step 4 Create the Database Link at `str1.net`

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

```
*/
CREATE DATABASE LINK str2.net CONNECT TO stradmin IDENTIFIED BY stradminpw
  USING 'str2.net';
/*
```

Step 5 Set Up Users at `str2.net`

Connect to `str2.net` as `SYS` user.

```
*/
CONNECT SYS/CHANGE_ON_INSTALL@str2.net AS SYSDBA
/*
```

Create the Streams administrator named `strmadmin` and grant this user the necessary privileges. These privileges enable the user to manage queues, execute subprograms in packages related to Streams, create rule sets, create rules, and monitor the Streams environment by querying data dictionary views and queue tables. In this example, the Streams administrator will be the apply user for the apply process and must be able to apply changes to the `hr.jobs` table at `str2.net`. Therefore, the Streams administrator is granted `ALL` privileges on this table. You may choose a different name for the Streams administrator.

Note:

- To ensure security, use a password other than `strmadminpw` for the Streams administrator.
 - The `SELECT_CATALOG_ROLE` is not required for the Streams administrator. It is granted in this example so that the Streams administrator can monitor the environment easily.
 - If you plan to use the Streams tool in Oracle Enterprise Manager, then grant the Streams administrator `SELECT ANY DICTIONARY` privilege, in addition to the privileges shown in this step.
 - The `ACCEPT` command must appear on a single line in the script.
-
-

See Also: ["Configuring a Streams Administrator"](#) on page 11-2

```
*/

GRANT CONNECT, RESOURCE, SELECT_CATALOG_ROLE
  TO strmadmin IDENTIFIED BY strmadminpw;

GRANT ALL ON hr.jobs TO strmadmin;

ACCEPT streams_tbs PROMPT 'Enter Streams administrator tablespace on str2.net: '

ALTER USER strmadmin DEFAULT TABLESPACE &streams_tbs
  QUOTA UNLIMITED ON &streams_tbs;

GRANT EXECUTE ON DBMS_APPLY_ADM          TO strmadmin;
GRANT EXECUTE ON DBMS_AQADM             TO strmadmin;
GRANT EXECUTE ON DBMS_STREAMS_ADM       TO strmadmin;
```

```

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

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

/*

```

Step 6 Set Up the Streams Queue at str2.net

Connect as the Streams administrator at str2.net.

```

*/

CONNECT stradmin/stradminpw@str2.net

/*

```

Run the `SET_UP_QUEUE` procedure to create a queue named `streams_queue` at `str2.net`. This queue will function as the Streams queue by holding the changes that will be applied at this database.

Running the `SET_UP_QUEUE` procedure performs the following actions:

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

```
*/  
  
EXEC DEMS_STREAMS_ADM.SET_UP_QUEUE();  
  
/*
```

Step 7 Check the Spool Results

Check the `streams_setup_simple.out` spool file to ensure that all actions finished successfully after this script is completed.

```
*/  
  
SET ECHO OFF  
SPOOL OFF  
  
/***** END OF SCRIPT *****/
```

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

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

1. [Show Output and Spool Results](#)
2. [Create an Alternate Tablespace for the LogMiner Tables at str1.net](#)
3. [Specify Supplemental Logging at str1.net](#)
4. [Configure Propagation at str1.net](#)
5. [Configure the Capture Process at str1.net](#)
6. [Instantiate the hr.jobs Table at str2.net](#)
7. [Drop the Supplemental Log Group for hr.jobs at str2.net](#)
8. [Configure the Apply Process at str2.net](#)
9. [Start the Apply Process at str2.net](#)
10. [Start the Capture Process at str1.net](#)
11. [Check the Spool Results](#)

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the next "END OF SCRIPT" line on page 21-18 into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to all of the databases in the environment.

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

Step 1 Show Output and Spool Results

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

```
*/

SET ECHO ON
SPOOL streams_share_jobs.out

/*
```

Step 2 Create an Alternate Tablespace for the LogMiner Tables at str1.net

By default, the LogMiner tables are in the SYSTEM tablespace, but the SYSTEM tablespace may not have enough space for these tables once a capture process starts to capture changes. Therefore, you must create an alternate tablespace for the LogMiner tables.

See Also: ["Alternate Tablespace for LogMiner Tables"](#) on page 2-20

Connect to str1.net as SYS user.

```
*/

CONNECT SYS/CHANGE_ON_INSTALL@str1.net AS SYSDBA

/*
```

Create an alternate tablespace for the LogMiner tables.

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

```
*/

ACCEPT tspace_name DEFAULT 'logmrts' PROMPT 'Enter tablespace name (for
example, logmrts): '

ACCEPT db_file_directory DEFAULT '' PROMPT 'Enter the complete path to the
datafile directory (for example, /usr/oracle/dbs): '

ACCEPT db_file_name DEFAULT 'logmrts.dbf' PROMPT 'Enter the name of the
datafile (for example, logmrts.dbf): '

CREATE TABLESPACE &tspace_name DATAFILE '&db_file_directory/&db_file_name'
  SIZE 25 M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;

EXECUTE DBMS_LOGMNR_D.SET_TABLESPACE('&tspace_name');

/*
```

Step 3 Specify Supplemental Logging at str1.net

Supplemental logging places additional information in the redo log for changes made to tables. The apply process needs this extra information to perform certain operations, such as unique row identification and conflict resolution. Because str1.net is the only database where changes are captured in this environment, it is the only database where you must specify supplemental logging for the hr.jobs table.

The following statement specifies an unconditional supplemental log group for the primary key column in the hr.jobs table.

See Also:

- ["Supplemental Logging in a Streams Environment"](#) on page 2-11
- ["Specifying Supplemental Logging at a Source Database"](#) on page 12-9

*/

```
ALTER TABLE hr.jobs ADD SUPPLEMENTAL LOG GROUP log_group_jobs_pk
(job_id) ALWAYS;
```

/*

Step 4 Configure Propagation at str1.net

Connect to str1.net as the stradmin user.

*/

```
CONNECT stradmin/stradminpw@str1.net
```

/*

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

*/

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES(
    table_name           => 'hr.jobs',
    streams_name         => 'str1_to_str2',
    source_queue_name    => 'stradmin.streams_queue',
    destination_queue_name => 'stradmin.streams_queue@str2.net',
    include_dml          => true,
    include_ddl          => true,
    source_database      => 'str1.net');
```

```
END;
```

```
/
```

/*

Step 5 Configure the Capture Process at str1.net

Configure the capture process to capture changes to the `hr.jobs` table at `str1.net`. This step specifies that changes to this table are captured by the capture process and enqueued into the specified queue.

```

*/

BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name      => 'hr.jobs',
    streams_type    => 'capture',
    streams_name    => 'capture_simp',
    queue_name      => 'strmadmin.streams_queue',
    include_dml     => true,
    include_ddl     => true);
END;
/

/*

```

Step 6 Instantiate the hr.jobs Table at str2.net

This example assumes that the `hr.jobs` table exists at both the `str1.net` database and the `str2.net` database, and that this table is identical at both databases. In this case, you can instantiate the table at the `str2.net` database by performing a metadata only export of the table at `str1.net` and then importing the resulting export dump file at `str2.net`. Performing this metadata only export/import records the instantiation SCN for the `hr.jobs` table at `str2.net`, which is required before an apply process at `str2.net` can apply changes to the table.

Open a different window and export the `hr.jobs` table at `str1.net` that will be instantiated at `str2.net`. Make sure you set the `OBJECT_CONSISTENT` export parameter to `y` and the `ROWS` export parameter to `n` when you run the export command. Also, make sure no DML or DDL changes are made to the `hr.jobs` table during the export.

The following is an example export command:

```
exp userid=hr/hr FILE=jobs_instant.dmp TABLES=jobs OBJECT_CONSISTENT=y ROWS=n
```

See Also: *Oracle9i Database Utilities* for information about performing an export

```
*/
```

```
PAUSE Press <RETURN> to continue when the export is complete in the other window that you opened.
```

```
/*
```

Transfer the export dump file `jobs_instant.dmp` to the destination database. In this example, the destination database is `str2.net`.

You can use binary FTP or some other method to transfer the export dump file to the destination database. You may need to open a different window to transfer the file.

```
*/
```

```
PAUSE Press <RETURN> to continue after transferring the dump file.
```

```
/*
```

In a different window, connect to the computer that runs the `str2.net` database and import the export dump file `jobs_instant.dmp` to instantiate the `jobs` table in the `str2.net` database. You can use telnet or remote login to connect to the computer that runs `str2.net`.

When you run the import command, make sure you set the `STREAMS_INSTANTIATION` import parameter to `y`. This parameter ensures that the import records export SCN information for each object imported.

The following is an example import command:

```
imp userid=hr/hr FILE=jobs_instant.dmp IGNORE=y COMMIT=y LOG=import.log  
STREAMS_INSTANTIATION=y
```

See Also: *Oracle9i Database Utilities* for information about performing an import

```
*/
```

```
PAUSE Press <RETURN> to continue after the import is complete at str2.net.
```

```
/*
```

Step 7 Drop the Supplemental Log Group for hr.jobs at str2.net

When you instantiated that hr.jobs table at str2.net, the supplemental log group from str1.net for the table was retained. This log group is not needed at str2.net because no capture process captures changes to this table at str2.net. You can remove the log group to avoid extraneous information in the redo log at str2.net. Connect to str2.net as the hr user.

```
*/  
  
CONNECT hr/hr@str2.net
```

```
/*
```

Drop the supplemental log group at str2.net.

```
*/  
  
ALTER TABLE hr.jobs DROP SUPPLEMENTAL LOG GROUP log_group_jobs_pk;
```

```
/*
```

Step 8 Configure the Apply Process at str2.net

Connect to str2.net as the strmadmin user.

```
*/  
  
CONNECT strmadmin/strmadminpw@str2.net
```

```
/*
```

Configure str2.net to apply changes to the hr.jobs table.

```
*/  
  
BEGIN  
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(  
    table_name      => 'hr.jobs',  
    streams_type    => 'apply',  
    streams_name    => 'apply_simp',  
    queue_name      => 'strmadmin.streams_queue',  
    include_dml     => true,  
    include_ddl     => true,  
    source_database => 'str1.net');  
END;  
/
```

```
/*
```

Step 9 Start the Apply Process at str2.net

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

```
*/
```

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

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

```
/*
```

Step 10 Start the Capture Process at str1.net

Connect to `str1.net` as the `stradmin` user.

```
*/
```

```
CONNECT stradmin/stradminpw@str1.net
```

```
/*
```

Start the capture process at `str1.net`.

```
*/
```

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

```
/*
```

Step 11 Check the Spool Results

Check the `streams_share_jobs.out` spool file to ensure that all actions finished successfully after this script is completed.

```
*/  
  
SET ECHO OFF  
SPOOL OFF  
  
/***** END OF SCRIPT *****/
```

Make Changes to the hr.jobs Table and View Results

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

Step 1 Make Changes to hr.jobs at str1.net

Make the following changes to the `hr.jobs` table.

```
CONNECT hr/hr@str1.net  
  
UPDATE hr.jobs SET max_salary=9545 WHERE job_id='PR_REP';  
COMMIT;  
  
ALTER TABLE hr.jobs ADD(duties VARCHAR2(4000));
```

Step 2 Query and Describe the hr.jobs Table at str2.net

After some time passes to allow for capture, propagation, and apply of the changes performed in the previous step, run the following query to confirm that the `UPDATE` change was propagated and applied at `str2.net`:

```
CONNECT hr/hr@str2.net  
  
SELECT * FROM hr.jobs WHERE job_id='PR_REP';
```

The value in the `max_salary` column should be 9545.

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

```
DESC hr.jobs
```

The `duties` column should be the last column.

Single Source Heterogeneous Replication Example

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

This chapter contains these topics:

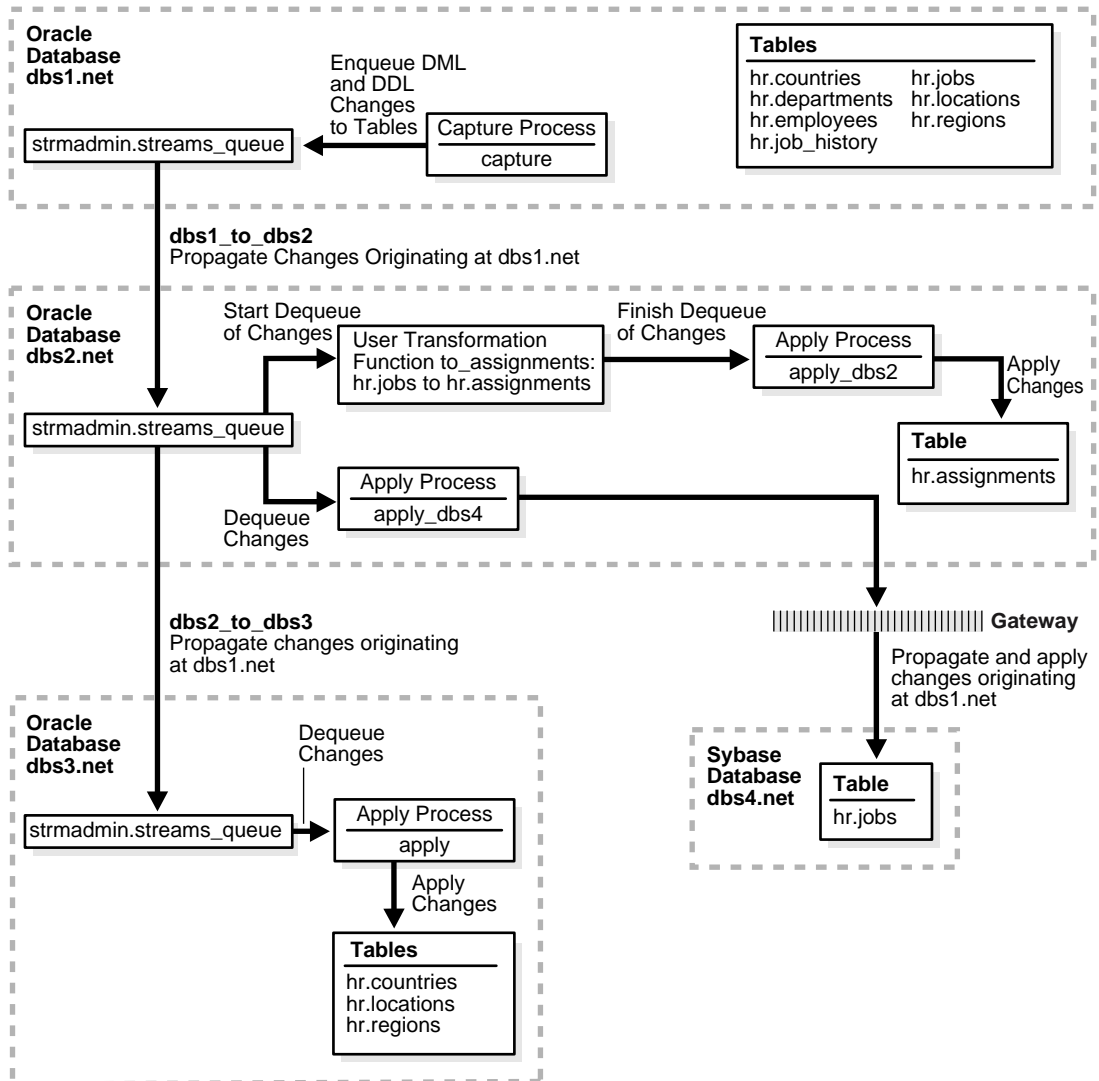
- [Overview of the Single Source Heterogeneous Replication Example](#)
- [Prerequisites](#)
- [Set Up Users and Create Queues and Database Links](#)
- [Example Scripts for Sharing Data from One Database](#)
- [Make DML and DDL Changes to Tables in the hr Schema](#)
- [Add Objects to an Existing Streams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

Overview of the Single Source Heterogeneous Replication Example

This example illustrates using Streams to replicate data between four databases. The environment is heterogeneous because three of the databases are Oracle databases and one is a Sybase database. DML and DDL changes made to tables in the `hr` schema at the `db1.net` Oracle database are captured and propagated to the other two Oracle databases. Only DML changes are captured and propagated to the `db4.net` database, because an apply process cannot apply DDL changes to a non-Oracle database. Changes to the `hr` schema occur only at `db1.net`. The `hr` schema is read-only at the other databases in the environment.

[Figure 22-1](#) provides an overview of the environment.

Figure 22–1 Example Environment That Shares Data from a Single Source Database



As illustrated in [Figure 22-1](#), `db1.net` contains the following tables in the `hr` schema:

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

This example uses directed networks, which means that captured changes at a source database are propagated to another database through one or more intermediate databases. Here, the `db1.net` database propagates changes to the `db3.net` database through the intermediate database `db2.net`. Also, the `db1.net` database propagates changes to the `db2.net` database, which applies the changes directly to the `db4.net` database through a gateway.

Some of the databases in the environment do not have certain tables. If the database is not an intermediate database for a table and the database does not contain the table, then changes to the table do not need to be propagated to that database. For example, the `departments`, `employees`, `job_history`, and `jobs` tables do not exist at `db3.net`. Therefore, `db2.net` does not propagate changes to these tables to `db3.net`.

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

1. The capture process captures DML and DDL changes for all of the tables in the `hr` schema and enqueues them into a queue at the `db1.net` database. In this example, changes to only four of the seven tables are propagated to destination databases, but in the example that illustrates "[Add Objects to an Existing Streams Replication Environment](#)" on page 22-64, the remaining tables in the `hr` schema are added to a destination database.
2. The `db1.net` database propagates these changes in the form of messages to a queue at `db2.net`.
3. At `db2.net`, DML changes to the `jobs` table are transformed into DML changes for the `assignments` table (which is a direct mapping of `jobs`) and then applied. Changes to other tables in the `hr` schema are not applied at `db2.net`.

4. Because the queue at `dbs3.net` receives changes from the queue at `dbs2.net` that originated in `countries`, `locations`, and `regions` tables at `dbs1.net`, these changes are propagated from `dbs2.net` to `dbs3.net`. This configuration is an example of directed networks.
5. The apply process at `dbs3.net` applies changes to the `countries`, `locations`, and `regions` tables.
6. Because `dbs4.net`, a Sybase database, receives changes from the queue at `dbs2.net` to the `jobs` table that originated at `dbs1.net`, these changes are applied remotely from `dbs2.net` using the `dbs4.net` database link through a gateway. This configuration is an example of heterogeneous support.

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:
 - `AQ_TM_PROCESSES`: This parameter establishes queue monitor processes. Values from 1 to 10 specify the number of queue monitor processes created to monitor the messages. If `AQ_TM_PROCESSES` is not specified or is set to 0, then the queue monitor processes are not created. In this example, `AQ_TM_PROCESSES` should be set to at least 1 at each database.

Setting the parameter to 1 or more starts the specified number of queue monitor processes. These queue monitor processes are responsible for managing time-based operations of messages such as delay and expiration, cleaning up retained messages after the specified retention time, and cleaning up consumed messages if the retention time is 0.
 - `GLOBAL_NAMES`: This parameter must be set to `true` at each database that is participating in your Streams environment.
 - `JOB_QUEUE_PROCESSES`: This parameter must be set to at least 2 at each database that is propagating events in your Streams environment. It should be set to the same value as the maximum number of jobs that can run simultaneously plus one. In this example, `dbs1.net` and `dbs2.net` propagate events. So, `JOB_QUEUE_PROCESSES` must be set to at least 2 at these databases.

- COMPATIBLE: This parameter must be set to 9.2.0 or higher.
- LOG_PARALLELISM: This parameter must be set to 1 at each database that captures events. In this example, this parameter must be set to 1 at `db1.net`.

See Also: ["Setting Initialization Parameters Relevant to Streams"](#) on page 11-4 for information about other initialization parameters that are important in a Streams environment

- Any database producing changes that will be captured must be running in ARCHIVELOG mode. In this example, changes are produced at `db1.net`, and so `db1.net` must be running in ARCHIVELOG mode.

See Also: *Oracle9i Database Administrator's Guide* for information about running a database in ARCHIVELOG mode

- Configure an Oracle gateway on `db2.net` to communicate with the Sybase database `db4.net`.

See Also: *Oracle9i Heterogeneous Connectivity Administrator's Guide*

- At the Sybase database `db4.net`, set up the `hr` user.

See Also: Your Sybase documentation for information about creating users and tables in your Sybase database

- Instantiate the `hr.jobs` table from the `db1.net` Oracle database at the `db4.net` Sybase database.

See Also: ["Instantiation in an Oracle to Non-Oracle Environment"](#) on page 9-7

- Configure your network and Oracle Net so that the following databases can communicate with each other:

- `db1.net` and `db2.net`
- `db2.net` and `db3.net`
- `db2.net` and `db4.net`

See Also: *Oracle9i Net Services Administrator's Guide*

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

Set Up Users and Create Queues and Database Links

Complete the following steps to set up users and create queues and database links for a Streams replication environment that includes three Oracle databases and one Sybase database.

1. [Show Output and Spool Results](#)
2. [Alter the hr.countries Table at dbs1.net](#)
3. [Set Up Users at dbs1.net](#)
4. [Create the Streams Queue at dbs1.net](#)
5. [Create the Database Link at dbs1.net](#)
6. [Set Up Users at dbs2.net](#)
7. [Create the Streams Queue at dbs2.net](#)
8. [Create the Database Links at dbs2.net](#)
9. [Create the hr.assignments Table at dbs2.net](#)
10. [Set Up Users at dbs3.net](#)
11. [Create the Streams Queue at dbs3.net](#)
12. [Drop All of the Tables in the hr Schema at dbs3.net](#)
13. [Check the Spool Results](#)

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the next "END OF SCRIPT" line on page 22-19 into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to all of the databases in the environment.

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

Step 1 Show Output and Spool Results

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

```
*/

SET ECHO ON
SPOOL streams_setup_single.out

/*
```

Step 2 Alter the hr.countries Table at dbs1.net

Connect to `dbs1.net` as the `hr` user.

```
*/

CONNECT hr/hr@dbs1.net

/*
```

Convert the `hr.countries` table from an index-organized table to a regular table. Currently, the capture process cannot capture changes to index-organized tables.

```
*/

ALTER TABLE countries RENAME TO countries_orig;

CREATE TABLE hr.countries(
  country_id      CHAR(2) CONSTRAINT country_id_nn_noiot NOT NULL,
  country_name    VARCHAR2(40),
  region_id       NUMBER,
  CONSTRAINT      country_c_id_pk_noiot PRIMARY KEY (country_id));
```



```
ALTER TABLE hr.countries
ADD (CONSTRAINT countr_reg_fk_noiot
      FOREIGN KEY (region_id)
      REFERENCES regions(region_id)) ;

INSERT INTO hr.countries (SELECT * FROM hr.countries_orig);

DROP TABLE hr.countries_orig CASCADE CONSTRAINTS;

ALTER TABLE locations
ADD (CONSTRAINT loc_c_id_fk
      FOREIGN KEY (country_id)
      REFERENCES countries(country_id));

/*
```

Step 3 Set Up Users at dbs1.net

Connect to `dbs1.net` as SYS user.

```
*/

CONNECT SYS/CHANGE_ON_INSTALL@dbs1.net AS SYSDBA

/*
```

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

Note:

- To ensure security, use a password other than `strmadminpw` for the Streams administrator.
 - The `SELECT_CATALOG_ROLE` is not required for the Streams administrator. It is granted in this example so that the Streams administrator can monitor the environment easily.
 - If you plan to use the Streams tool in Oracle Enterprise Manager, then grant the Streams administrator `SELECT ANY DICTIONARY` privilege, in addition to the privileges shown in this step.
 - The `ACCEPT` command must appear on a single line in the script.
-
-

See Also: ["Configuring a Streams Administrator"](#) on page 11-2

```
*/

GRANT CONNECT, RESOURCE, SELECT_CATALOG_ROLE
  TO strmadmin IDENTIFIED BY strmadminpw;

ACCEPT streams_tbs PROMPT 'Enter Streams administrator tablespace on dbs1.net: '

ALTER USER strmadmin DEFAULT TABLESPACE &streams_tbs
  QUOTA UNLIMITED ON &streams_tbs;

GRANT EXECUTE ON DBMS_AQADM           TO strmadmin;
GRANT EXECUTE ON DBMS_CAPTURE_ADM    TO strmadmin;
GRANT EXECUTE ON DBMS_FLASHBACK      TO strmadmin;
GRANT EXECUTE ON DBMS_PROPAGATION_ADM TO strmadmin;
GRANT EXECUTE ON DBMS_STREAMS_ADM    TO strmadmin;
```

```

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

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

/*

```

Step 4 Create the Streams Queue at dbs1.net

Connect as the Streams administrator at the database where you want to capture changes. In this example, that database is `dbs1.net`.

```

*/

CONNECT strmadmin/strmadminpw@dbs1.net

/*

```

Run the `SET_UP_QUEUE` procedure to create a queue named `streams_queue` at `dbs1.net`. This queue will function as the Streams queue by holding the captured changes that will be propagated to other databases.

Running the `SET_UP_QUEUE` procedure performs the following actions:

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

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

Step 5 Create the Database Link at dbs1.net

Create the database link from the database where changes are captured to the database where changes are propagated. In this example, the database where changes are captured is `dbs1.net`, and these changes are propagated to `dbs2.net`.

```
*/  
  
CREATE DATABASE LINK dbs2.net CONNECT TO strmadmin IDENTIFIED BY strmadminpw  
    USING 'dbs2.net';  
  
/*
```

Step 6 Set Up Users at dbs2.net

Connect to `dbs2.net` as SYS user.

```
*/  
  
CONNECT SYS/CHANGE_ON_INSTALL@dbs2.net AS SYSDBA  
  
/*
```

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

Note:

- To ensure security, use a password other than `strmadminpw` for the Streams administrator.
- The `SELECT_CATALOG_ROLE` is not required for the Streams administrator. It is granted in this example so that the Streams administrator can monitor the environment easily.
- If you plan to use the Streams tool in Oracle Enterprise Manager, then grant the Streams administrator `SELECT ANY DICTIONARY` privilege, in addition to the privileges shown in this step.
- The `ACCEPT` command must appear on a single line in the script.

See Also: ["Configuring a Streams Administrator"](#) on page 11-2

```

*/
GRANT CONNECT, RESOURCE, SELECT_CATALOG_ROLE
  TO strmadmin IDENTIFIED BY strmadminpw;

ACCEPT streams_tbs PROMPT 'Enter Streams administrator tablespace on dbs2.net: '

ALTER USER strmadmin DEFAULT TABLESPACE &streams_tbs
  QUOTA UNLIMITED ON &streams_tbs;

GRANT EXECUTE ON DBMS_APPLY_ADM          TO strmadmin;
GRANT EXECUTE ON DBMS_AQADM             TO strmadmin;
GRANT EXECUTE ON DBMS_PROPAGATION_ADM   TO strmadmin;
GRANT EXECUTE ON DBMS_STREAMS_ADM       TO strmadmin;

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

```

```
BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_RULE_OBJ,
    grantee      => 'stradmin',
    grant_option => FALSE);
END;
/

/*
```

Step 7 Create the Streams Queue at dbs2.net

Connect as the Streams administrator at `dbs2.net`.

```
*/

CONNECT stradmin/stradminpw@dbs2.net

/*
```

Run the `SET_UP_QUEUE` procedure to create a queue named `streams_queue` at `dbs2.net`. This queue will function as the Streams queue by holding the changes that will be applied at this database and the changes that will be propagated to other databases.

Running the `SET_UP_QUEUE` procedure performs the following actions:

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

```
*/

EXEC DBMS_STREAMS_ADM.SET_UP_QUEUE();

/*
```

Step 8 Create the Database Links at dbs2.net

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

Note: On some non-Oracle databases, including Sybase, you must ensure that the characters in the username and password are in the correct case. Therefore, double quotation marks are specified for the username and password at the Sybase database.

```
*/
```

```
CREATE DATABASE LINK dbs3.net CONNECT TO strmadmin IDENTIFIED BY strmadminpw  
    USING 'dbs3.net';
```

```
CREATE DATABASE LINK dbs4.net CONNECT TO "hr" IDENTIFIED BY "hrpass"  
    USING 'dbs4.net';
```

```
/*
```

Step 9 Create the hr.assignments Table at dbs2.net

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

Connect as `hr` at `dbs2.net`.

```
*/
```

```
CONNECT hr/hr@dbs2.net
```

```
/*
```

Create the `hr.assignments` table in the `dbs2.net` database.

```
*/  
  
CREATE TABLE hr.assignments AS SELECT * FROM hr.jobs;  
  
ALTER TABLE hr.assignments ADD PRIMARY KEY (job_id);  
  
/*
```

Step 10 Set Up Users at `dbs3.net`

Connect to `dbs3.net` as SYS user.

```
*/  
  
CONNECT SYS/CHANGE_ON_INSTALL@dbs3.net AS SYSDBA  
  
/*
```

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

Note:

- To ensure security, use a password other than `strmadminpw` for the Streams administrator.
 - The `SELECT_CATALOG_ROLE` is not required for the Streams administrator. It is granted in this example so that the Streams administrator can monitor the environment easily.
 - If you plan to use the Streams tool in Oracle Enterprise Manager, then grant the Streams administrator `SELECT ANY DICTIONARY` privilege, in addition to the privileges shown in this step.
 - The `ACCEPT` command must appear on a single line in the script.
-
-

See Also: ["Configuring a Streams Administrator"](#) on page 11-2


```

*/

GRANT CONNECT, RESOURCE, SELECT_CATALOG_ROLE
  TO strmadmin IDENTIFIED BY strmadminpw;

ACCEPT streams_tbs PROMPT 'Enter Streams administrator tablespace on dbs3.net: '

ALTER USER strmadmin DEFAULT TABLESPACE &streams_tbs
  QUOTA UNLIMITED ON &streams_tbs;

GRANT EXECUTE ON DBMS_APPLY_ADM      TO strmadmin;
GRANT EXECUTE ON DBMS_AQADM          TO strmadmin;
GRANT EXECUTE ON DBMS_STREAMS_ADM    TO strmadmin;

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

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

/*

```

Step 11 Create the Streams Queue at dbs3.net

Connect as the Streams administrator at dbs3.net.

```

*/

CONNECT strmadmin/strmadminpw@dbs3.net

/*

```

Run the `SET_UP_QUEUE` procedure to create a queue named `streams_queue` at `dbs3.net`. This queue will function as the Streams queue by holding the changes that will be applied at this database.

Running the `SET_UP_QUEUE` procedure performs the following actions:

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

```
*/
```

```
EXEC DEMS_STREAMS_ADM.SET_UP_QUEUE();
```

```
/*
```

Step 12 Drop All of the Tables in the `hr` Schema at `db3.net`

This example illustrates instantiating tables in the `hr` schema by exporting them from `db1.net` and importing them into `db3.net`. You must delete these tables at `db3.net` for the instantiation portion of this example to work properly.

Connect as `hr` at `db3.net`.

```
*/
```

```
CONNECT hr/hr@db3.net
```

```
/*
```

Drop all tables in the `hr` schema in the `db3.net` database.

Attention: If you complete this step and drop all of the tables in the `hr` schema, then you should complete the remaining sections of this example to reinstantiate the `hr` schema at `db3.net`. If the `hr` schema does not exist in an Oracle database, then some examples in the Oracle documentation set may fail.

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

```
/*
```

Step 13 Check the Spool Results

Check the `streams_setup_single.out` spool file to ensure that all actions finished successfully after this script is completed.

```
*/
```

```
SET ECHO OFF  
SPOOL OFF
```

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

Example Scripts for Sharing Data from One Database

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

- ["Simple Configuration for Sharing Data from a Single Database"](#) on page 22-21 demonstrates a simple way to configure the environment. This example uses the `DBMS_STREAMS_ADM` package to create a capture process, propagations, and apply processes, as well as the rule sets associated with them. Using the `DBMS_STREAMS_ADM` package is the simplest way to configure a Streams environment.
- ["Flexible Configuration for Sharing Data from a Single Database"](#) on page 22-40 demonstrates a more flexible way to configure this environment. This example uses the `DBMS_CAPTURE_ADM` package to create a capture process, the `DBMS_PROPAGATION_ADM` package to create propagations, and the `DBMS_APPLY_ADM` package to create apply processes. Also, this example uses the `DBMS_RULES_ADM` package to create and populate the rule sets associated with these capture processes, propagations, and apply processes. Using these packages, instead of the `DBMS_STREAMS_ADM` package, provides more configuration options and flexibility.

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

Simple Configuration for Sharing Data from a Single Database

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

1. [Show Output and Spool Results](#)
2. [Create an Alternate Tablespace for the LogMiner Tables at dbs1.net](#)
3. [Specify Supplemental Logging at dbs1.net](#)
4. [Configure Propagation at dbs1.net](#)
5. [Configure the Capture Process at dbs1.net](#)
6. [Set the Instantiation SCN for the Existing Tables at Other Databases](#)
7. [Instantiate the dbs1.net Tables at dbs3.net](#)
8. [Drop the Supplemental Log Groups at dbs3.net](#)
9. [Configure the Apply Process at dbs3.net](#)
10. [Specify hr as the Apply User for the Apply Process at dbs3.net](#)
11. [Grant the hr User Execute Privilege on the Apply Process Rule Set](#)
12. [Start the Apply Process at dbs3.net](#)
13. [Configure Propagation at dbs2.net](#)
14. [Create the Rule-Based Transformation for Row LCRs at dbs2.net](#)
15. [Configure the Apply Process for Local Apply at dbs2.net](#)
16. [Specify hr as the Apply User for the Apply Process at dbs2.net](#)
17. [Grant the hr User Execute Privilege on the Apply Process Rule Set](#)
18. [Start the Apply Process at dbs2.net for Local Apply](#)
19. [Configure the Apply Process at dbs2.net for Apply at dbs4.net](#)
20. [Start the Apply Process at dbs2.net for Apply at dbs4.net](#)
21. [Start the Capture Process at dbs1.net](#)
22. [Check the Spool Results](#)

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the next "END OF SCRIPT" line on page 22-40 into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to all of the databases in the environment.

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

Step 1 Show Output and Spool Results

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

```
*/  
  
SET ECHO ON  
SPOOL streams_share_schema1.out  
  
/*
```

Step 2 Create an Alternate Tablespace for the LogMiner Tables at dbs1.net

By default, the LogMiner tables are in the `SYSTEM` tablespace, but the `SYSTEM` tablespace may not have enough space for these tables once a capture process starts to capture changes. Therefore, you must create an alternate tablespace for the LogMiner tables.

See Also: ["Alternate Tablespace for LogMiner Tables"](#) on page 2-20

Connect to `dbs1.net` as `SYS` user.

```
*/  
  
CONNECT SYS/CHANGE_ON_INSTALL@dbs1.net AS SYSDBA  
  
/*
```

Create an alternate tablespace for the LogMiner tables.

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

```
*/

ACCEPT tspace_name DEFAULT 'logmrts' PROMPT 'Enter the name of the tablespace
(for example, logmrts): '

ACCEPT db_file_directory DEFAULT '' PROMPT 'Enter the complete path to the
datafile directory (for example, /usr/oracle/dbs): '

ACCEPT db_file_name DEFAULT 'logmrts.dbf' PROMPT 'Enter the name of the
datafile (for example, logmrts.dbf): '

CREATE TABLESPACE &tspace_name DATAFILE '&db_file_directory/&db_file_name'
      SIZE 25 M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;

EXECUTE DBMS_LOGMNR_D.SET_TABLESPACE('&tspace_name');

/*
```

Step 3 Specify Supplemental Logging at dbs1.net

Supplemental logging places additional information in the redo log for changes made to tables. The apply process needs this extra information to perform certain operations, such as unique row identification and conflict resolution. Because `dbs1.net` is the only database where changes are captured in this environment, it is the only database where you must specify supplemental logging for the tables in the `hr` schema.

Specify an unconditional supplemental log group for all primary key columns in the `hr` schema.

See Also:

- ["Supplemental Logging in a Streams Environment"](#) on page 2-11
- ["Specifying Supplemental Logging at a Source Database"](#) on page 12-9

```
*/  
  
ALTER TABLE hr.countries ADD SUPPLEMENTAL LOG GROUP log_group_countries_pk  
    (country_id) ALWAYS;  
  
ALTER TABLE hr.departments ADD SUPPLEMENTAL LOG GROUP log_group_departments_pk  
    (department_id) ALWAYS;  
  
ALTER TABLE hr.employees ADD SUPPLEMENTAL LOG GROUP log_group_employees_pk  
    (employee_id) ALWAYS;  
  
ALTER TABLE hr.jobs ADD SUPPLEMENTAL LOG GROUP log_group_jobs_pk  
    (job_id) ALWAYS;  
  
ALTER TABLE hr.job_history ADD SUPPLEMENTAL LOG GROUP log_group_job_history_pk  
    (employee_id, start_date) ALWAYS;  
  
ALTER TABLE hr.locations ADD SUPPLEMENTAL LOG GROUP log_group_locations_pk  
    (location_id) ALWAYS;  
  
ALTER TABLE hr.regions ADD SUPPLEMENTAL LOG GROUP log_group_regions_pk  
    (region_id) ALWAYS;  
  
/*
```

Step 4 Configure Propagation at dbs1.net
Connect to dbs1.net as the strmadmin user.

```
*/  
  
CONNECT strmadmin/strmadminpw@dbs1.net  
  
/*
```


Configure and schedule propagation of DML and DDL changes in the `hr` schema from the queue at `db1.net` to the queue at `db2.net`.

```

*/

BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES(
    schema_name          => 'hr',
    streams_name         => 'db1_to_db2',
    source_queue_name    => 'strmadmin.streams_queue',
    destination_queue_name => 'strmadmin.streams_queue@db2.net',
    include_dml         => true,
    include_ddl         => true,
    source_database      => 'db1.net');
END;
/

/*

```

Step 5 Configure the Capture Process at `db1.net`

Configure the capture process to capture changes to the entire `hr` schema at `db1.net`. This step specifies that changes to the tables in the specified schema are captured by the capture process and enqueued into the specified queue.

```

*/

BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
    schema_name      => 'hr',
    streams_type     => 'capture',
    streams_name     => 'capture',
    queue_name       => 'strmadmin.streams_queue',
    include_dml     => true,
    include_ddl     => true);
END;
/

/*

```

Step 6 Set the Instantiation SCN for the Existing Tables at Other Databases

In this example, the `hr.jobs` table already exists at `dbs2.net` and `dbs4.net`. At `dbs2.net`, this table is named `assignments`, but it has the same shape and data as the `jobs` table at `dbs1.net`. Also, in this example, `dbs4.net` is a Sybase database. All of the other tables in the Streams environment are instantiated at the other databases using Export/Import.

Because the `hr.jobs` table already exists at `dbs2.net` and `dbs4.net`, this example uses the `GET_SYSTEM_CHANGE_NUMBER` function in the `DBMS_FLASHBACK` package at `dbs1.net` to obtain the current SCN for the database. This SCN is used at `dbs2.net` to run the `SET_TABLE_INSTANTIATION_SCN` procedure in the `DBMS_APPLY_ADM` package. Running this procedure twice sets the instantiation SCN for the `hr.jobs` table at `dbs2.net` and `dbs4.net`.

The `SET_TABLE_INSTANTIATION_SCN` procedure controls which LCRs for a table are ignored by an apply process and which LCRs for a table are applied by an apply process. If the commit SCN of an LCR for a table from a source database is less than or equal to the instantiation SCN for that table at a destination database, then the apply process at the destination database discards the LCR. Otherwise, the apply process applies the LCR.

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

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

```

*/

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

/*

```

Step 7 Instantiate the dbs1.net Tables at dbs3.net

Open a different window and export the tables at `dbs1.net` that will be instantiated at `dbs3.net`. Make sure you set the `OBJECT_CONSISTENT` export parameter to `y` when you run the export command. Also, make sure no DDL changes are made to the objects being exported during the export.

The following is an example export command:

```
exp userid=hr/hr FILE=hr_instant1.dmp TABLES=countries,locations,regions
OBJECT_CONSISTENT=y
```

See Also: *Oracle9i Database Utilities* for information about performing an export

```

*/

PAUSE Press <RETURN> to continue when the export is complete in the other window
that you opened.

/*

```

Transfer the export dump file `hr_instant1.dmp` to the destination database. In this example, the destination database is `dbs3.net`.

You can use binary FTP or some other method to transfer the export dump file to the destination database. You may need to open a different window to transfer the file.

*/

PAUSE Press <RETURN> to continue after transferring the dump file.

/*

In a different window, connect to the computer that runs the `dbs3.net` database and import the export dump file `hr_instant1.dmp` to instantiate the `countries`, `locations`, and `regions` tables in the `dbs3.net` database. You can use telnet or remote login to connect to the computer that runs `dbs3.net`.

When you run the import command, make sure you set the `STREAMS_INSTANTIATION` import parameter to `y`. This parameter ensures that the import records export SCN information for each object imported.

The following is an example import command:

```
imp userid=hr/hr FILE=hr_instant1.dmp IGNORE=y FULL=y COMMIT=y LOG=import.log
STREAMS_INSTANTIATION=y
```

See Also: *Oracle9i Database Utilities* for information about performing an import

*/

PAUSE Press <RETURN> to continue after the import is complete at `dbs3.net`.

/*

Step 8 Drop the Supplemental Log Groups at `dbs3.net`

When you instantiated the `hr` schema at `dbs3.net`, the supplemental log groups from `dbs1.net` were retained. These log groups are not needed at `dbs3.net` because no capture process captures changes to the tables in the `hr` schema at `dbs3.net`. You can remove the log groups to avoid extraneous information in the redo log at `dbs3.net`.

Connect to `db3.net` as the `hr` user.

```
*/  
  
CONNECT hr/hr@db3.net
```

```
/*
```

Drop the supplemental log groups at `db3.net`.

```
*/  
  
ALTER TABLE hr.countries DROP SUPPLEMENTAL LOG GROUP log_group_countries_pk;  
  
ALTER TABLE hr.locations DROP SUPPLEMENTAL LOG GROUP log_group_locations_pk;  
  
ALTER TABLE hr.regions DROP SUPPLEMENTAL LOG GROUP log_group_regions_pk;
```

```
/*
```

Step 9 Configure the Apply Process at `db3.net`

Connect to `db3.net` as the `strmadmin` user.

```
*/  
  
CONNECT strmadmin/strmadminpw@db3.net
```

```
/*
```

Configure `db3.net` to apply changes to the `countries` table, `locations` table, and `regions` table.

```
*/  
  
BEGIN  
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(  
    table_name      => 'hr.countries',  
    streams_type    => 'apply',  
    streams_name    => 'apply',  
    queue_name      => 'strmadmin.streams_queue',  
    include_dml     => true,  
    include_ddl     => true,  
    source_database => 'db3.net');  
END;  
/
```

```

BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name      => 'hr.locations',
    streams_type    => 'apply',
    streams_name    => 'apply',
    queue_name      => 'strmadmin.streams_queue',
    include_dml     => true,
    include_ddl     => true,
    source_database => 'dbs1.net');
END;
/

BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name      => 'hr.regions',
    streams_type    => 'apply',
    streams_name    => 'apply',
    queue_name      => 'strmadmin.streams_queue',
    include_dml     => true,
    include_ddl     => true,
    source_database => 'dbs1.net');
END;
/

/*

```

Step 10 Specify hr as the Apply User for the Apply Process at dbs3.net

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

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

See Also: ["Configuring a Streams Administrator"](#) on page 11-2

```
*/

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

/*
```

Step 11 Grant the hr User Execute Privilege on the Apply Process Rule Set

Because the hr user was specified as the apply user in the previous step, the hr user requires execute privilege on the rule set used by the apply process

```
*/

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

/*
```

Step 12 Start the Apply Process at dbs3.net

Set the `disable_on_error` parameter to `n` so that the apply process will not be disabled if it encounters an error, and start the apply process at `dbs3.net`.

```
*/

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

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

/*
```

Step 13 Configure Propagation at dbs2.net

Connect to `dbs2.net` as the `strmadmin` user.

```
*/

CONNECT strmadmin/strmadminpw@dbs2.net

/*
```

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


```
*/  
  
BEGIN  
  DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES(  
    table_name           => 'hr.countries',  
    streams_name         => 'dbs2_to_dbs3',  
    source_queue_name    => 'strmadmin.streams_queue',  
    destination_queue_name => 'strmadmin.streams_queue@dbs3.net',  
    include_dml          => true,  
    include_ddl          => true,  
    source_database      => 'dbs1.net');  
END;  
/  
  
BEGIN  
  DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES(  
    table_name           => 'hr.locations',  
    streams_name         => 'dbs2_to_dbs3',  
    source_queue_name    => 'strmadmin.streams_queue',  
    destination_queue_name => 'strmadmin.streams_queue@dbs3.net',  
    include_dml          => true,  
    include_ddl          => true,  
    source_database      => 'dbs1.net');  
END;  
/  
  
BEGIN  
  DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES(  
    table_name           => 'hr.regions',  
    streams_name         => 'dbs2_to_dbs3',  
    source_queue_name    => 'strmadmin.streams_queue',  
    destination_queue_name => 'strmadmin.streams_queue@dbs3.net',  
    include_dml          => true,  
    include_ddl          => true,  
    source_database      => 'dbs1.net');  
END;  
/  
  
/*
```

Step 14 Create the Rule-Based Transformation for Row LCRs at dbs2.net

Connect to `dbs2.net` as the `hr` user.

```
*/  
  
CONNECT hr/hr@dbs2.net  
  
/*
```

Create the rule-based transformation function that transforms row changes resulting from DML statements to the `jobs` table from `dbs1.net` into row changes to the `assignments` table on `dbs2.net`.

The following function transforms every row LCR for the `jobs` table into a row LCR for the `assignments` table.

Note: If DDL changes were also applied to the `assignments` table, then another transformation would be required for the DDL LCRs. This transformation would need to change the object name and the DDL text.

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

Step 15 Configure the Apply Process for Local Apply at dbs2.net

Connect to `dbs2.net` as the `strmadmin` user.

```
*/
CONNECT strmadmin/strmadminpw@dbs2.net
/*
```

Configure `dbs2.net` to apply changes to the `assignments` table. Remember that the `assignments` table receives changes from the `jobs` table at `dbs1.net`.

```
*/
DECLARE
  to_assignments_rulename_dml  VARCHAR2(30);
  dummy_rule                   VARCHAR2(30);
  action_ctx_dml               SYS.RE$NV_LIST;
  ac_name                       VARCHAR2(30) := 'STREAMS$_TRANSFORM_FUNCTION';
BEGIN
  -- DML changes to the jobs table from dbs1.net are applied to the assignments
  -- table. The to_assignments_rulename_dml variable is an out parameter
  -- in this call.
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name      => 'hr.jobs', -- jobs, not assignments, specified
    streams_type    => 'apply',
    streams_name    => 'apply_dbs2',
    queue_name      => 'strmadmin.streams_queue',
    include_dml     => true,
    include_ddl     => false,
    source_database => 'dbs1.net',
    dml_rule_name   => to_assignments_rulename_dml,
    ddl_rule_name   => dummy_rule);
  -- Specify the name-value pair in the action context
  action_ctx_dml := SYS.RE$NV_LIST(SYS.RE$NV_ARRAY());
  action_ctx_dml.ADD_PAIR(
    ac_name,
    SYS.ANYDATA.CONVERTVARCHAR2('hr.to_assignments_trans_dml'));
  -- Modify the rule for jobs to use the transformation.
  DBMS_RULE_ADM.ALTER_RULE(
    rule_name       => to_assignments_rulename_dml,
    action_context  => action_ctx_dml);
END;
/
```

/*

Step 16 Specify hr as the Apply User for the Apply Process at dbs2.net

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

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

See Also: ["Configuring a Streams Administrator"](#) on page 11-2

*/

```
BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name => 'apply_dbs2',
    apply_user => 'hr');
END;
/

/*
```

Step 17 Grant the hr User Execute Privilege on the Apply Process Rule Set

Because the `hr` user was specified as the apply user in the previous step, the `hr` user requires execute privilege on the rule set used by the apply process

```

*/

DECLARE
    rs_name VARCHAR2(64); -- Variable to hold rule set name
BEGIN
    SELECT RULE_SET_OWNER||'.'||RULE_SET_NAME
        INTO rs_name
        FROM DBA_APPLY
        WHERE APPLY_NAME='APPLY_DBS2';
    DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE(
        privilege => SYS.DBMS_RULE_ADM.EXECUTE_ON_RULE_SET,
        object_name => rs_name,
        grantee => 'hr');
END;
/

/*

```

Step 18 Start the Apply Process at dbs2.net for Local Apply

Set the `disable_on_error` parameter to `n` so that the apply process will not be disabled if it encounters an error, and start the apply process for local apply at `dbs2.net`.

```

*/

BEGIN
    DBMS_APPLY_ADM.SET_PARAMETER(
        apply_name => 'apply_dbs2',
        parameter => 'disable_on_error',
        value => 'n');
END;
/

BEGIN
    DBMS_APPLY_ADM.START_APPLY(
        apply_name => 'apply_dbs2');
END;
/

/*

```

Step 19 Configure the Apply Process at dbs2.net for Apply at dbs4.net

Configure the apply process for `dbs4.net`, which is a Sybase database. The `dbs2.net` database is acting as a gateway to `dbs4.net`. Therefore, the apply process for `dbs4.net` must be configured at `dbs2.net`. The apply process cannot apply DDL changes to non-Oracle databases. Therefore, the `include_ddl` parameter is set to `false` when the `ADD_TABLE_RULES` procedure is run.

```

*/

BEGIN
  DBMS_APPLY_ADM.CREATE_APPLY(
    queue_name      => 'strmadmin.streams_queue',
    apply_name      => 'apply_dbs4',
    apply_database_link => 'dbs4.net',
    apply_captured  => true);
END;
/

BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name      => 'hr.jobs',
    streams_type    => 'apply',
    streams_name    => 'apply_dbs4',
    queue_name      => 'strmadmin.streams_queue',
    include_dml     => true,
    include_ddl     => false,
    source_database => 'dbs1.net');
END;
/

/*

```

Step 20 Start the Apply Process at dbs2.net for Apply at dbs4.net

Set the `disable_on_error` parameter to `n` so that the apply process will not be disabled if it encounters an error, and start the remote apply for Sybase using database link `dbs4.net`.

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

Step 21 Start the Capture Process at dbs1.net

Connect to `dbs1.net` as the `strmadmin` user.

```
*/  
  
CONNECT strmadmin/strmadminpw@dbs1.net  
  
/*
```

Start the capture process at `dbs1.net`.

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

Step 22 Check the Spool Results

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

```
*/  
  
SET ECHO OFF  
SPOOL OFF  
  
/*
```

You can now make DML and DDL changes to specific tables at `db1.net` and see these changes replicated to the other databases in the environment based on the rules you configured for the Streams processes and propagations in this environment.

See Also: ["Make DML and DDL Changes to Tables in the hr Schema"](#) on page 22-62 for examples of changes that are replicated in this environment

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

Flexible Configuration for Sharing Data from a Single Database

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

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

Note: Neither the `ALL_STREAMS_TABLE_RULES` nor the `DBA_STREAMS_TABLE_RULES` data dictionary view is populated by the rules created in this example. To view the rules created in this example, you must query the `ALL_RULES`, `DBA_RULES`, or `USER_RULES` data dictionary view.

This example includes the following steps:

1. Show Output and Spool Results
2. Create an Alternate Tablespace for the LogMiner Tables at dbs1.net
3. Specify Supplemental Logging at dbs1.net
4. Configure Propagation at dbs1.net
5. Configure the Capture Process at dbs1.net
6. Prepare the hr Schema at dbs1.net for Instantiation
7. Set the Instantiation SCN for the Existing Tables at Other Databases
8. Instantiate the dbs1.net Tables at dbs3.net
9. Drop the Supplemental Log Groups at dbs3.net
10. Configure the Apply Process at dbs3.net
11. Grant the hr User Execute Privilege on the Apply Process Rule Set
12. Start the Apply Process at dbs3.net
13. Configure Propagation at dbs2.net
14. Create the Rule-Based Transformation for Row LCRs at dbs2.net
15. Configure the Apply Process for Local Apply at dbs2.net
16. Grant the hr User Execute Privilege on the Apply Process Rule Set
17. Start the Apply Process at dbs2.net for Local Apply
18. Configure the Apply Process at dbs2.net for Apply at dbs4.net
19. Start the Apply Process at dbs2.net for Apply at dbs4.net
20. Start the Capture Process at dbs1.net
21. Check the Spool Results

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the next "END OF SCRIPT" line on page 22-61 into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to all of the databases in the environment.

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

Step 1 Show Output and Spool Results

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

```
*/  
  
SET ECHO ON  
SPOOL streams_share_schema2.out
```

```
/*
```

Step 2 Create an Alternate Tablespace for the LogMiner Tables at dbs1.net

By default, the LogMiner tables are in the `SYSTEM` tablespace, but the `SYSTEM` tablespace may not have enough space for these tables once a capture process starts to capture changes. Therefore, you must create an alternate tablespace for the LogMiner tables.

See Also: ["Alternate Tablespace for LogMiner Tables"](#) on page 2-20

Connect to `dbs1.net` as `SYS` user.

```
*/  
  
CONNECT SYS/CHANGE_ON_INSTALL@dbs1.net AS SYSDBA
```

```
/*
```

Create an alternate tablespace for the LogMiner tables.

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

```
*/

ACCEPT tspace_name DEFAULT 'logmrts' PROMPT 'Enter the name of the tablespace
(for example, logmrts): '

ACCEPT db_file_directory DEFAULT '' PROMPT 'Enter the complete path to the
datafile directory (for example, /usr/oracle/dbs): '

ACCEPT db_file_name DEFAULT 'logmrts.dbf' PROMPT 'Enter the name of the
datafile (for example, logmrts.dbf): '

CREATE TABLESPACE &tspace_name DATAFILE '&db_file_directory/&db_file_name'
    SIZE 25 M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;

EXECUTE DBMS_LOGMNR_D.SET_TABLESPACE('&tspace_name');

/*
```

Step 3 Specify Supplemental Logging at dbs1.net

Supplemental logging places additional information in the redo log for changes made to tables. The apply process needs this extra information to perform certain operations, such as unique row identification and conflict resolution. Because `dbs1.net` is the only database where changes are captured in this environment, it is the only database where you must specify supplemental logging for the tables in the `hr` schema.

Specify an unconditional supplemental log group for all primary key columns in the `hr` schema.

See Also:

- ["Supplemental Logging in a Streams Environment"](#) on page 2-11
- ["Specifying Supplemental Logging at a Source Database"](#) on page 12-9

```
*/  
  
ALTER TABLE hr.countries ADD SUPPLEMENTAL LOG GROUP log_group_countries_pk  
    (country_id) ALWAYS;  
  
ALTER TABLE hr.departments ADD SUPPLEMENTAL LOG GROUP log_group_departments_pk  
    (department_id) ALWAYS;  
  
ALTER TABLE hr.employees ADD SUPPLEMENTAL LOG GROUP log_group_employees_pk  
    (employee_id) ALWAYS;  
  
ALTER TABLE hr.jobs ADD SUPPLEMENTAL LOG GROUP log_group_jobs_pk  
    (job_id) ALWAYS;  
  
ALTER TABLE hr.job_history ADD SUPPLEMENTAL LOG GROUP log_group_job_history_pk  
    (employee_id, start_date) ALWAYS;  
  
ALTER TABLE hr.locations ADD SUPPLEMENTAL LOG GROUP log_group_locations_pk  
    (location_id) ALWAYS;  
  
ALTER TABLE hr.regions ADD SUPPLEMENTAL LOG GROUP log_group_regions_pk  
    (region_id) ALWAYS;  
  
/*
```

Step 4 Configure Propagation at dbs1.net Connect to dbs1.net as the strmadmin user.

```
*/  
  
CONNECT strmadmin/strmadminpw@dbs1.net  
  
/*
```

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

```

*/

BEGIN
  -- Create the rule set
  DBMS_RULE_ADM.CREATE_RULE_SET(
    rule_set_name      => 'strmadmin.propagation_dbs1_rules',
    evaluation_context => 'SYS.STREAMS$_EVALUATION_CONTEXT');
  -- Create rules for all modifications to the hr schema
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name => 'strmadmin.all_hr_dml',
    condition => ' :dml.get_object_owner() = 'HR' AND ' ||
                ' :dml.is_null_tag() = 'Y' AND ' ||
                ' :dml.get_source_database_name() = 'DBS1.NET' ');
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name => 'strmadmin.all_hr_ddl',
    condition => ' :ddl.get_object_owner() = 'HR' AND ' ||
                ' :ddl.is_null_tag() = 'Y' AND ' ||
                ' :ddl.get_source_database_name() = 'DBS1.NET' ');
  -- Add rules to rule set
  DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'strmadmin.all_hr_dml',
    rule_set_name => 'strmadmin.propagation_dbs1_rules');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'strmadmin.all_hr_ddl',
    rule_set_name => 'strmadmin.propagation_dbs1_rules');
  -- Create the propagation
  DBMS_PROPAGATION_ADM.CREATE_PROPAGATION(
    propagation_name => 'dbs1_to_dbs2',
    source_queue      => 'strmadmin.streams_queue',
    destination_queue => 'strmadmin.streams_queue',
    destination_dblink => 'dbs2.net',
    rule_set_name     => 'strmadmin.propagation_dbs1_rules');
END;
/

/*

```

Step 5 Configure the Capture Process at dbs1.net

Create a capture process and rules to capture the entire hr schema at dbs1.net.

```

*/

BEGIN
  -- Create the rule set
  DBMS_RULE_ADM.CREATE_RULE_SET(
    rule_set_name      => 'strmadmin.demo_rules',
    evaluation_context => 'SYS.STREAMS$_EVALUATION_CONTEXT');
  -- Create rules that specify the entire hr schema
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name => 'strmadmin.schema_hr_dml',
    condition => ' :dml.get_object_owner() = 'HR' AND ' ||
                ' :dml.is_null_tag() = 'Y' AND ' ||
                ' :dml.get_source_database_name() = 'DBS1.NET' ');
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name => 'strmadmin.schema_hr_ddl',
    condition => ' :ddl.get_object_owner() = 'HR' AND ' ||
                ' :ddl.is_null_tag() = 'Y' AND ' ||
                ' :ddl.get_source_database_name() = 'DBS1.NET' ');
  -- Add the rules to the rule set
  DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'strmadmin.schema_hr_dml',
    rule_set_name => 'strmadmin.demo_rules');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'strmadmin.schema_hr_ddl',
    rule_set_name => 'strmadmin.demo_rules');
  -- Create a capture process that uses the rule set
  DBMS_CAPTURE_ADM.CREATE_CAPTURE(
    queue_name      => 'strmadmin.streams_queue',
    capture_name    => 'capture',
    rule_set_name   => 'strmadmin.demo_rules');
END;
/

/*

```

Step 6 Prepare the hr Schema at dbs1.net for Instantiation

While still connected as the Streams administrator at dbs1.net, prepare the hr schema at dbs1.net for instantiation at dbs3.net. This step marks the lowest SCN of the tables in the schema for instantiation. SCNs subsequent to the lowest SCN can be used for instantiation.

Note: This step is not required in the ["Simple Configuration for Sharing Data from a Single Database"](#) on page 22-21. In that example, when the `ADD_SCHEMA_RULES` procedure in the `DBMS_STREAMS_ADM` package is run in Step 5, the `PREPARE_SCHEMA_INSTANTIATION` procedure in the `DBMS_CAPTURE_ADM` package is run automatically for the `hr` schema.

```

*/

BEGIN
  DBMS_CAPTURE_ADM.PREPARE_SCHEMA_INSTANTIATION(
    schema_name => 'hr' );
END;
/

/*

```

Step 7 Set the Instantiation SCN for the Existing Tables at Other Databases

In this example, the `hr.jobs` table already exists at `dbs2.net` and `dbs4.net`. At `dbs2.net`, this table is named `assignments`, but it has the same shape and data as the `jobs` table at `dbs1.net`. Also, in this example, `dbs4.net` is a Sybase database. All of the other tables in the Streams environment are instantiated at the other databases using Export/Import.

Because the `hr.jobs` table already exists at `dbs2.net` and `dbs4.net`, this example uses the `GET_SYSTEM_CHANGE_NUMBER` function in the `DBMS_FLASHBACK` package at `dbs1.net` to obtain the current SCN for the database. This SCN is used at `dbs2.net` to run the `SET_TABLE_INSTANTIATION_SCN` procedure in the `DBMS_APPLY_ADM` package. Running this procedure twice sets the instantiation SCN for the `hr.jobs` table at `dbs2.net` and `dbs4.net`.

The `SET_TABLE_INSTANTIATION_SCN` procedure controls which LCRs for a table are ignored by an apply process and which LCRs for a table are applied by an apply process. If the commit SCN of an LCR for a table from a source database is less than or equal to the instantiation SCN for that table at a destination database, then the apply process at the destination database discards the LCR. Otherwise, the apply process applies the LCR.

In this example, both of the apply processes at `db3.net` will apply transactions to the `hr.jobs` table with SCNs that were committed after SCN obtained in this step.

Note: This example assumes that the contents of the `hr.jobs` table at `db1.net`, `db2.net` (as `hr.assignments`), and `db4.net` are consistent when you complete this step. You may want to lock the table at each database while you complete this step to ensure consistency.

```

*/

DECLARE
    iscn NUMBER;          -- Variable to hold instantiation SCN value
BEGIN
    iscn := DEMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER();
    DEMS_APPLY_ADM.SET_TABLE_INSTANTIATION_SCN@DBS2.NET(
        source_object_name => 'hr.jobs',
        source_database_name => 'db1.net',
        instantiation_scn => iscn);
    DEMS_APPLY_ADM.SET_TABLE_INSTANTIATION_SCN@DBS2.NET(
        source_object_name => 'hr.jobs',
        source_database_name => 'db1.net',
        instantiation_scn => iscn,
        apply_database_link => 'db4.net');
END;
/

/*

```

Step 8 Instantiate the `db1.net` Tables at `db3.net`

Open a different window and export the tables at `db1.net` that will be instantiated at `db3.net`. Make sure you set the `OBJECT_CONSISTENT` export parameter to `y` when you run the export command. Also, make sure no DDL changes are made to the objects being exported during the export.

The following is an example export command:

```

exp userid=hr/hr FILE=hr_instant1.dmp TABLES=countries,locations,regions
OBJECT_CONSISTENT=y

```


See Also: *Oracle9i Database Utilities* for information about performing an export

```
*/
```

PAUSE Press <RETURN> to continue when the export is complete in the other window that you opened.

```
/*
```

Transfer the export dump file `hr_instant1.dmp` to the destination database. In this example, the destination database is `dbs3.net`.

You can use binary FTP or some other method to transfer the export dump file to the destination database. You may need to open a different window to transfer the file.

```
*/
```

PAUSE Press <RETURN> to continue after transferring the dump file.

```
/*
```

In a different window, connect to the computer that runs the `dbs3.net` database and import the export dump file `hr_instant1.dmp` to instantiate the `countries`, `locations`, and `regions` tables in the `dbs3.net` database. You can use telnet or remote login to connect to the computer that runs `dbs3.net`.

When you run the import command, make sure you set the `STREAMS_INSTANTIATION` import parameter to `y`. This parameter ensures that the import records export SCN information for each object imported.

The following is an example import command:

```
imp userid=hr/hr FILE=hr_instant1.dmp IGNORE=y FULL=y COMMIT=y LOG=import.log  
STREAMS_INSTANTIATION=y
```

See Also: *Oracle9i Database Utilities* for information about performing an import

```
*/  
  
PAUSE Press <RETURN> to continue after the import is complete at dbs3.net.  
  
/*
```

Step 9 Drop the Supplemental Log Groups at dbs3.net

When you instantiated the `hr` schema at `dbs3.net`, the supplemental log groups from `dbs1.net` were retained. These log groups are not needed at `dbs3.net` because no capture process captures changes to the tables in the `hr` schema at `dbs3.net`. You can remove the log groups to avoid extraneous information in the redo log at `dbs3.net`.

Connect to `dbs3.net` as the `hr` user.

```
*/  
  
CONNECT hr/hr@dbs3.net  
  
/*
```

Drop the supplemental log groups at `dbs3.net`.

```
*/  
  
ALTER TABLE hr.countries DROP SUPPLEMENTAL LOG GROUP log_group_countries_pk;  
  
ALTER TABLE hr.locations DROP SUPPLEMENTAL LOG GROUP log_group_locations_pk;  
  
ALTER TABLE hr.regions DROP SUPPLEMENTAL LOG GROUP log_group_regions_pk;  
  
/*
```

Step 10 Configure the Apply Process at dbs3.net

Connect to `dbs3.net` as the `strmadmin` user.

```
*/  
  
CONNECT strmadmin/strmadminpw@dbs3.net  
  
/*
```

Configure `dbs3.net` to apply DML and DDL changes to the `countries` table, `locations` table, and `regions` table.

```

*/

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

```

```

DBMS_RULE_ADM.CREATE_RULE(
  rule_name      => 'strmadmin.all_regions_ddl',
  condition      => ' :ddl.get_object_owner() = 'HR' AND ' ||
                  ' :ddl.get_object_name() = 'REGIONS' AND ' ||
                  ' :ddl.is_null_tag() = 'Y' AND ' ||
                  ' :ddl.get_source_database_name() = 'DBS1.NET' ');
-- Add rules to rule set
DBMS_RULE_ADM.ADD_RULE(
  rule_name      => 'strmadmin.all_countries_dml',
  rule_set_name  => 'strmadmin.apply_rules');
DBMS_RULE_ADM.ADD_RULE(
  rule_name      => 'strmadmin.all_countries_ddl',
  rule_set_name  => 'strmadmin.apply_rules');
DBMS_RULE_ADM.ADD_RULE(
  rule_name      => 'strmadmin.all_locations_dml',
  rule_set_name  => 'strmadmin.apply_rules');
DBMS_RULE_ADM.ADD_RULE(
  rule_name      => 'strmadmin.all_locations_ddl',
  rule_set_name  => 'strmadmin.apply_rules');
DBMS_RULE_ADM.ADD_RULE(
  rule_name      => 'strmadmin.all_regions_dml',
  rule_set_name  => 'strmadmin.apply_rules');
DBMS_RULE_ADM.ADD_RULE(
  rule_name      => 'strmadmin.all_regions_ddl',
  rule_set_name  => 'strmadmin.apply_rules');
-- Create the apply process
DBMS_APPLY_ADM.CREATE_APPLY(
  queue_name     => 'strmadmin.streams_queue',
  apply_name     => 'apply',
  rule_set_name  => 'strmadmin.apply_rules',
  apply_user     => 'hr',
  apply_captured => true);
END;
/

/*

```

Step 11 Grant the hr User Execute Privilege on the Apply Process Rule Set

Because the `hr` user was specified as the apply user in the previous step, the `hr` user requires execute privilege on the rule set used by the apply process

```

*/

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

/*

```

Step 12 Start the Apply Process at dbs3.net

Set the `disable_on_error` parameter to `n` so that the apply process will not be disabled if it encounters an error, and start the apply process at `dbs3.net`.

```

*/

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

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

/*

```

Step 13 Configure Propagation at dbs2.net

Connect to `dbs2.net` as the `strmadmin` user.

```
*/
CONNECT strmadmin/strmadminpw@dbs2.net
/*
```

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

```
*/
BEGIN
  -- Create the rule set
  DBMS_RULE_ADM.CREATE_RULE_SET(
    rule_set_name      => 'strmadmin.propagation_dbs3_rules',
    evaluation_context => 'SYS.STREAMS$_EVALUATION_CONTEXT');
  -- Create rules for all modifications to the countries table
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name => 'strmadmin.all_countries_dml',
    condition => ' :dml.get_object_owner() = ''HR'' AND ' ||
                 ' :dml.get_object_name() = ''COUNTRIES'' AND ' ||
                 ' :dml.is_null_tag() = ''Y'' AND ' ||
                 ' :dml.get_source_database_name() = ''DBS1.NET'' ');
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name => 'strmadmin.all_countries_ddl',
    condition => ' :ddl.get_object_owner() = ''HR'' AND ' ||
                 ' :ddl.get_object_name() = ''COUNTRIES'' AND ' ||
                 ' :ddl.is_null_tag() = ''Y'' AND ' ||
                 ' :ddl.get_source_database_name() = ''DBS1.NET'' ');
  -- Create rules for all modifications to the locations table
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name => 'strmadmin.all_locations_dml',
    condition => ' :dml.get_object_owner() = ''HR'' AND ' ||
                 ' :dml.get_object_name() = ''LOCATIONS'' AND ' ||
                 ' :dml.is_null_tag() = ''Y'' AND ' ||
                 ' :dml.get_source_database_name() = ''DBS1.NET'' ');
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name => 'strmadmin.all_locations_ddl',
    condition => ' :ddl.get_object_owner() = ''HR'' AND ' ||
                 ' :ddl.get_object_name() = ''LOCATIONS'' AND ' ||
                 ' :ddl.is_null_tag() = ''Y'' AND ' ||
                 ' :ddl.get_source_database_name() = ''DBS1.NET'' ');

```

```

-- Create rules for all modifications to the regions table
DBMS_RULE_ADM.CREATE_RULE(
  rule_name    => 'strmadmin.all_regions_dml',
  condition    => ' :dml.get_object_owner() = 'HR' AND ' ||
                ' :dml.get_object_name() = 'REGIONS' AND ' ||
                ' :dml.is_null_tag() = 'Y' AND ' ||
                ' :dml.get_source_database_name() = 'DBS1.NET' ' ');
DBMS_RULE_ADM.CREATE_RULE(
  rule_name    => 'strmadmin.all_regions_ddl',
  condition    => ' :ddl.get_object_owner() = 'HR' AND ' ||
                ' :ddl.get_object_name() = 'REGIONS' AND ' ||
                ' :ddl.is_null_tag() = 'Y' AND ' ||
                ' :ddl.get_source_database_name() = 'DBS1.NET' ' ');
-- Add rules to rule set
DBMS_RULE_ADM.ADD_RULE(
  rule_name      => 'strmadmin.all_countries_dml',
  rule_set_name  => 'strmadmin.propagation_dbs3_rules');
DBMS_RULE_ADM.ADD_RULE(
  rule_name      => 'strmadmin.all_countries_ddl',
  rule_set_name  => 'strmadmin.propagation_dbs3_rules');
DBMS_RULE_ADM.ADD_RULE(
  rule_name      => 'strmadmin.all_locations_dml',
  rule_set_name  => 'strmadmin.propagation_dbs3_rules');
DBMS_RULE_ADM.ADD_RULE(
  rule_name      => 'strmadmin.all_locations_ddl',
  rule_set_name  => 'strmadmin.propagation_dbs3_rules');
DBMS_RULE_ADM.ADD_RULE(
  rule_name      => 'strmadmin.all_regions_dml',
  rule_set_name  => 'strmadmin.propagation_dbs3_rules');
DBMS_RULE_ADM.ADD_RULE(
  rule_name      => 'strmadmin.all_regions_ddl',
  rule_set_name  => 'strmadmin.propagation_dbs3_rules');
-- Create the propagation
DBMS_PROPAGATION_ADM.CREATE_PROPAGATION(
  propagation_name => 'dbs2_to_dbs3',
  source_queue     => 'strmadmin.streams_queue',
  destination_queue => 'strmadmin.streams_queue',
  destination_dblink => 'dbs3.net',
  rule_set_name    => 'strmadmin.propagation_dbs3_rules');
END;
/
/*

```

Step 14 Create the Rule-Based Transformation for Row LCRs at dbs2.net

Connect to `dbs2.net` as the `hr` user.

```
*/  
  
CONNECT hr/hr@dbs2.net  
  
/*
```

Create the rule-based transformation function that transforms row changes resulting from DML statements to the `jobs` table from `dbs1.net` into row changes to the `assignments` table on `dbs2.net`.

The following function transforms every row LCR for the `jobs` table into a row LCR for the `assignments` table.

Note: If DDL changes were also applied to the `assignments` table, then another transformation would be required for the DDL LCRs. This transformation would need to change the object name and the DDL text.

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


Step 15 Configure the Apply Process for Local Apply at dbs2.net

Connect to `dbs2.net` as the `strmadmin` user.

```
*/
CONNECT strmadmin/strmadminpw@dbs2.net
/*
```

Configure `dbs2.net` to apply changes to the local `assignments` table. Remember that the `assignments` table receives changes from the `jobs` table at `dbs1.net`.

```
*/
DECLARE
  action_ctx_dml      SYS.RE$NV_LIST;
  action_ctx_ddl      SYS.RE$NV_LIST;
  ac_name             VARCHAR2(30) := 'STREAMS$_TRANSFORM_FUNCTION';
BEGIN
  -- Specify the name-value pair in the action context
  action_ctx_dml := SYS.RE$NV_LIST(SYS.RE$NV_ARRAY());
  action_ctx_dml.ADD_PAIR(
    ac_name,
    SYS.ANYDATA.CONVERTVARCHAR2('hr.to_assignments_trans_dml'));
  -- Create the rule set strmadmin.apply_rules
  DBMS_RULE_ADM.CREATE_RULE_SET(
    rule_set_name      => 'strmadmin.apply_rules',
    evaluation_context => 'SYS.STREAMS$_EVALUATION_CONTEXT');
  -- Create a rule that transforms all DML changes to the jobs table into
  -- DML changes for assignments table
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name          => 'strmadmin.all_jobs_dml',
    condition           => ':dml.get_object_owner() = ''HR'' AND ' ||
                          ':dml.get_object_name() = ''JOBS'' AND ' ||
                          ':dml.is_null_tag() = ''Y'' AND ' ||
                          ':dml.get_source_database_name() = ''DBS1.NET'' ',
    action_context     => action_ctx_dml);
  -- Add the rule to the rule set
  DBMS_RULE_ADM.ADD_RULE(
    rule_name          => 'strmadmin.all_jobs_dml',
    rule_set_name     => 'strmadmin.apply_rules');
```

```
-- Create an apply process that uses the rule set
DBMS_APPLY_ADM.CREATE_APPLY(
  queue_name      => 'strmadmin.streams_queue',
  apply_name      => 'apply_dbs2',
  rule_set_name   => 'strmadmin.apply_rules',
  apply_user      => 'hr',
  apply_captured => true);
END;
/

/*
```

Step 16 Grant the hr User Execute Privilege on the Apply Process Rule Set

Because the hr user was specified as the apply user in the previous step, the hr user requires execute privilege on the rule set used by the apply process

```
*/

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

/*
```

Step 17 Start the Apply Process at dbs2.net for Local Apply

Set the `disable_on_error` parameter to `n` so that the apply process will not be disabled if it encounters an error, and start the apply process for local apply at `dbs2.net`.

```
*/

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

```

BEGIN
    DBMS_APPLY_ADM.START_APPLY(
        apply_name => 'apply_dbs2');
END;
/

/*

```

Step 18 Configure the Apply Process at dbs2.net for Apply at dbs4.net

Configure `dbs2.net` to apply DML changes to the `jobs` table at `dbs4.net`, which is a Sybase database. Remember that these changes originated at `dbs1.net`.

```

*/

BEGIN
    -- Create the rule set
    DBMS_RULE_ADM.CREATE_RULE_SET(
        rule_set_name      => 'strmadmin.apply_dbs4_rules',
        evaluation_context => 'SYS.STREAMS$EVALUATION_CONTEXT');
    -- Create rule strmadmin.all_jobs_remote for all modifications
    -- to the jobs table
    DBMS_RULE_ADM.CREATE_RULE(
        rule_name      => 'strmadmin.all_jobs_remote',
        condition      => ' :dml.get_object_owner() = ''HR'' AND ' ||
                        ' :dml.get_object_name() = ''JOBS'' AND ' ||
                        ' :dml.is_null_tag() = ''Y'' AND ' ||
                        ' :dml.get_source_database_name() = ''DBS1.NET'' ');
    -- Add the rule to the rule set
    DBMS_RULE_ADM.ADD_RULE(
        rule_name      => 'strmadmin.all_jobs_remote',
        rule_set_name => 'strmadmin.apply_dbs4_rules');
    -- Create an apply process that uses the rule set
    DBMS_APPLY_ADM.CREATE_APPLY(
        queue_name      => 'strmadmin.streams_queue',
        apply_name      => 'apply_dbs4',
        rule_set_name   => 'strmadmin.apply_dbs4_rules',
        apply_database_link => 'dbs4.net',
        apply_captured  => true);
END;
/

/*

```

Step 19 Start the Apply Process at dbs2.net for Apply at dbs4.net

Set the `disable_on_error` parameter to `n` so that the apply process will not be disabled if it encounters an error, and start the remote apply for Sybase using database link `dbs4.net`.

```
*/

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

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

/*
```

Step 20 Start the Capture Process at dbs1.net

Connect to `dbs1.net` as the `strmadmin` user.

```
*/

CONNECT strmadmin/strmadminpw@dbs1.net

/*
```

Start the capture process at `dbs1.net`.

```
*/

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

/*
```

Step 21 Check the Spool Results

Check the `streams_share_schema2.out` spool file to ensure that all actions finished successfully after this script is completed.

```
*/
```

```
SET ECHO OFF  
SPOOL OFF
```

```
/*
```

You can now make DML and DDL changes to specific tables at `dbsl.net` and see these changes replicated to the other databases in the environment based on the rules you configured for the Streams processes and propagations in this environment.

See Also: ["Make DML and DDL Changes to Tables in the hr Schema"](#) on page 22-62 for examples of changes that are replicated in this environment

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

Make DML and DDL Changes to Tables in the hr Schema

After completing either of the examples described in the ["Example Scripts for Sharing Data from One Database"](#) section on page 22-20, you can make DML and DDL changes to the tables in the `hr` schema at the `dbs1.net` database. These changes will be replicated to the other databases in the environment based on the rules you configured for Streams processes and propagations. You can check the other databases to see that the changes have been replicated.

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

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

Step 1 Make DML and DDL Changes to Tables in the hr Schema

Make the following changes:

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

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

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

```
ALTER TABLE hr.locations RENAME COLUMN state_province TO state_or_province;
```

Step 2 Query the hr.assignments Table at dbs2.net

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

```
CONNECT hr/hr@dbs2.net
```

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

You should see 10000 for the value of the max_salary.

Step 3 Query and Describe the hr.locations Table at dbs3.net

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

```
CONNECT hr/hr@dbs3.net
```

```
SELECT * FROM hr.locations WHERE location_id=3300;
```

You should see the row inserted into the hr.locations table at dbs1.net in the previous step.

Next, describe the hr.locations table at to confirm that the ALTER TABLE change was propagated and applied correctly.

```
DESC hr.locations
```

The fifth column in the table should be state_or_province.

Add Objects to an Existing Streams Replication Environment

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

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

- `departments`
- `employees`
- `job_history`
- `jobs`

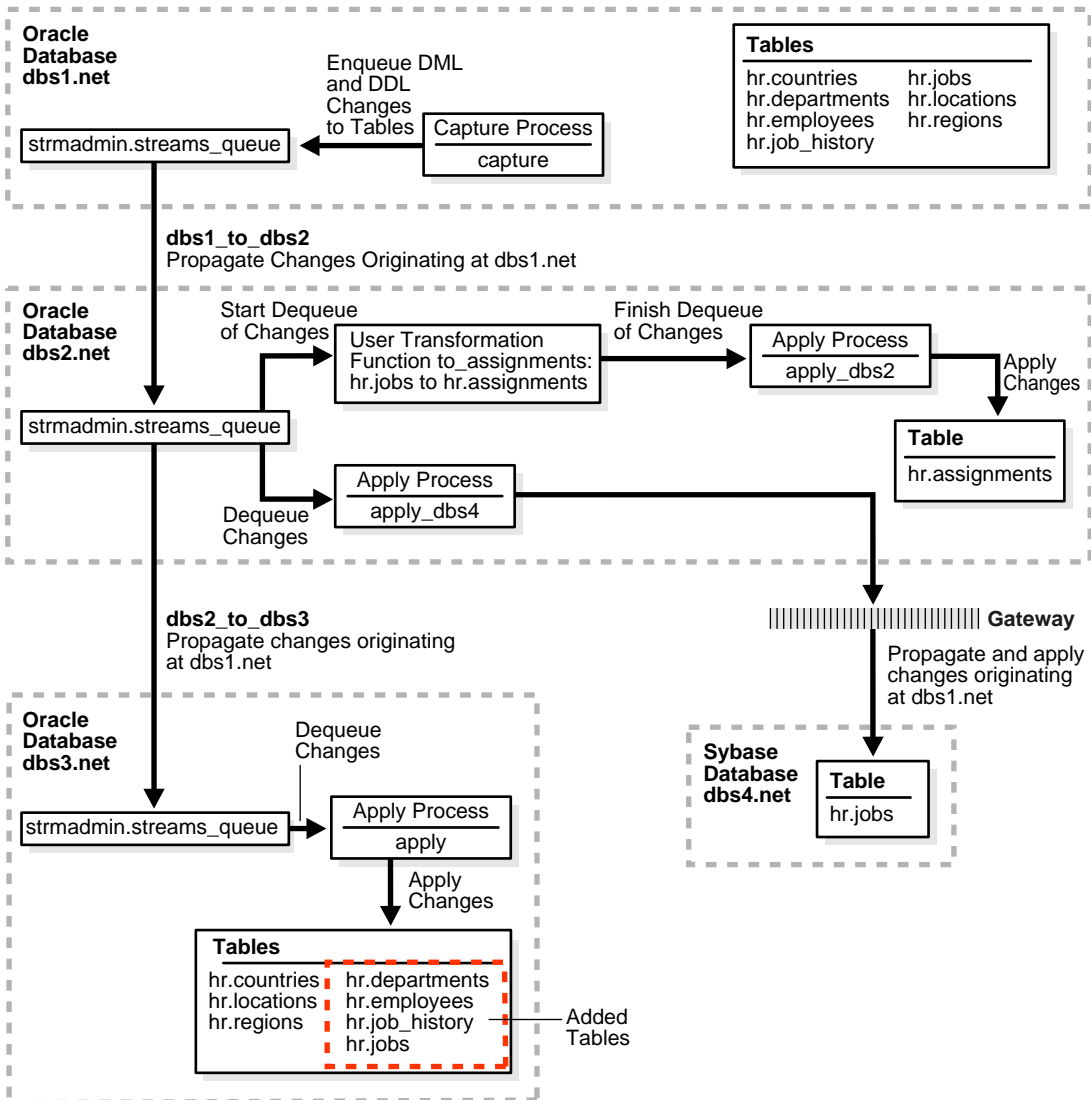
When you complete this example, Streams processes changes to these tables with the following series of actions:

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

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

[Figure 22-2](#) provides an overview of the environment with the added tables.

Figure 22–2 Adding Objects to db3.net in the Environment



Complete the following steps to replicate these tables to the `db3.net` database.

1. [Show Output and Spool Results](#)
2. [Stop the Apply Process at db3.net](#)
3. [Configure the Apply Process for the Added Tables at db3.net](#)
4. [Specify the Table Propagation Rules for the Added Tables at db2.net](#)
5. [Prepare the Four Added Tables for Instantiation at db1.net](#)
6. [Instantiate the db1.net Tables at db3.net](#)
7. [Drop the Supplemental Log Groups at db3.net](#)
8. [Start the Apply Process at db3.net](#)
9. [Check the Spool Results](#)

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the next "END OF SCRIPT" line on page 22-74 into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to all of the databases in the environment.

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

Step 1 Show Output and Spool Results

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

```
*/  
  
SET ECHO ON  
SPOOL streams_addobjs.out  
  
/*
```

Step 2 Stop the Apply Process at db3.net

Until you finish adding objects to `db3.net`, you must ensure that the apply process that will apply changes for the added objects does not try to apply changes

for these objects. You can do this by stopping the capture process at the source database. Or, you can do this by stopping propagation of changes from `db32.net` to `db33.net`. Yet another alternative is to stop the apply process at `db33.net`. This example stops the apply process at `db33.net`.

Connect to `db33.net` as the `strmadmin` user.

```
*/  
  
CONNECT strmadmin/strmadminpw@db33.net  
  
/*
```

Stop the apply process at `db33.net`.

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

Step 3 Configure the Apply Process for the Added Tables at `db33.net`

Configure the apply process at `db33.net` to apply changes to the tables you are adding.

```
*/  
  
BEGIN  
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(  
    table_name      => 'hr.departments',  
    streams_type    => 'apply',  
    streams_name    => 'apply',  
    queue_name      => 'strmadmin.streams_queue',  
    include_dml     => true,  
    include_ddl     => true,  
    source_database => 'db31.net');  
END;  
/  
  
/*
```

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name      => 'hr.employees',
    streams_type    => 'apply',
    streams_name    => 'apply',
    queue_name      => 'strmadmin.streams_queue',
    include_dml     => true,
    include_ddl     => true,
    source_database => 'dbs1.net');
END;
/

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

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

/*
```

Step 4 Specify the Table Propagation Rules for the Added Tables at dbs2.net

Connect to `dbs2.net` as the `strmadmin` user.

```
*/  
  
CONNECT strmadmin/strmadminpw@dbs2.net  
  
/*
```

Add the tables to the rules for propagation from the queue at `dbs2.net` to the queue at `dbs3.net`.

```
*/  
  
BEGIN  
  DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES(  
    table_name           => 'hr.departments',  
    streams_name         => 'dbs2_to_dbs3',  
    source_queue_name    => 'strmadmin.streams_queue',  
    destination_queue_name => 'strmadmin.streams_queue@dbs3.net',  
    include_dml          => true,  
    include_ddl          => true,  
    source_database      => 'dbs1.net');  
END;  
/  
  
BEGIN  
  DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES(  
    table_name           => 'hr.employees',  
    streams_name         => 'dbs2_to_dbs3',  
    source_queue_name    => 'strmadmin.streams_queue',  
    destination_queue_name => 'strmadmin.streams_queue@dbs3.net',  
    include_dml          => true,  
    include_ddl          => true,  
    source_database      => 'dbs1.net');  
END;  
/
```

```

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

BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES(
    table_name           => 'hr.jobs',
    streams_name         => 'dbs2_to_dbs3',
    source_queue_name    => 'strmadmin.streams_queue',
    destination_queue_name => 'strmadmin.streams_queue@dbs3.net',
    include_dml          => true,
    include_ddl          => true,
    source_database      => 'dbs1.net');
END;
/

/*

```

Step 5 Prepare the Four Added Tables for Instantiation at dbs1.net

Connect to `dbs1.net` as the `strmadmin` user.

```

*/

CONNECT strmadmin/strmadminpw@dbs1.net

/*

```

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

See Also:

- ["Preparing Database Objects for Instantiation at a Source Database" on page 12-11](#)
- ["Streams Data Dictionary for Propagations" on page 3-25](#)
- ["Streams Data Dictionary for an Apply Process" on page 4-33](#)

```

*/

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

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

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

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

/*

```

Step 6 Instantiate the dbs1.net Tables at dbs3.net

Open a different window and export the tables at `dbs1.net` that will be instantiated at `dbs3.net`. Make sure you set the `OBJECT_CONSISTENT` export parameter to `y` when you run the export command. Also, make sure no DDL changes are made to the objects being exported during the export.

The following is an example export command:

```
exp userid=hr/hr FILE=hr_instant2.dmp  
TABLES=departments,employees,job_history,jobs OBJECT_CONSISTENT=y
```

See Also: *Oracle9i Database Utilities* for information about performing an export

*/

PAUSE Press <RETURN> to continue when the export is complete in the other window that you opened.

/*

Transfer the export dump file `hr_instant2.dmp` to the destination database. In this example, the destination database is `dbs3.net`.

You can use binary FTP or some other method to transfer the export dump file to the destination database. You may need to open a different window to transfer the file.

*/

PAUSE Press <RETURN> to continue after transferring the dump file.

/*

In a different window, connect to the computer that runs the `dbs3.net` database and import the export dump file `hr_instant2.dmp` to instantiate the tables in the `dbs3.net` database. You can use telnet or remote login to connect to the computer that runs `dbs3.net`.

When you run the import command, make sure you set the `STREAMS_INSTANTIATION` import parameter to `y`. This parameter ensures that the import records export SCN information for each object imported.

The following is an example import command:

```
imp userid=hr/hr FILE=hr_instant2.dmp IGNORE=y FULL=y COMMIT=y LOG=import.log  
STREAMS_INSTANTIATION=y
```

See Also: *Oracle9i Database Utilities* for information about performing an import


```
*/
```

```
PAUSE Press <RETURN> to continue after the import is complete at dbs3.net.
```

```
/*
```

Step 7 Drop the Supplemental Log Groups at dbs3.net

When you instantiated the hr schema at dbs3.net, the supplemental log groups from dbs1.net were retained. These log groups are not needed at dbs3.net because no capture process captures changes to the tables in the hr schema at dbs3.net. You can remove the log groups to avoid extraneous information in the redo log at dbs3.net.

Connect to dbs3.net as the hr user.

```
*/
```

```
CONNECT hr/hr@dbs3.net
```

```
/*
```

Drop the supplemental log groups at dbs3.net.

```
*/
```

```
ALTER TABLE hr.departments DROP SUPPLEMENTAL LOG GROUP log_group_departments_pk;
```

```
ALTER TABLE hr.employees DROP SUPPLEMENTAL LOG GROUP log_group_employees_pk;
```

```
ALTER TABLE hr.jobs DROP SUPPLEMENTAL LOG GROUP log_group_jobs_pk;
```

```
ALTER TABLE hr.job_history DROP SUPPLEMENTAL LOG GROUP log_group_job_history_pk;
```

```
/*
```

Step 8 Start the Apply Process at dbs3.net

Start the apply process at dbs3.net. This apply process was stopped in Step 2.

Connect to dbs3.net as the strmadmin user.

```
*/
```

```
CONNECT strmadmin/strmadminpw@dbs3.net
```

```
/*
```

Start the apply process at dbs3.net.

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

Step 9 Check the Spool Results

Check the `streams_addobjs.out` spool file to ensure that all actions finished successfully after this script is completed.

```
*/  
  
SET ECHO OFF  
SPOOL OFF  
  
/***** END OF SCRIPT *****/
```

Make a DML Change to the hr.employees Table

After completing the examples described in the "[Add Objects to an Existing Streams Replication Environment](#)" section on page 22-64, you can make DML and DDL changes to the tables in the `hr` schema at the `dbs1.net` database. These changes will be replicated to `dbs3.net`. You can check these tables at `dbs3.net` to see that the changes have been replicated.

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

Step 1 Make a DML Change to the hr.employees Table

Make the following change:

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

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

Step 2 Query the hr.employees Table at dbs3.net

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

```
CONNECT hr/hr@dbs3.net
```

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

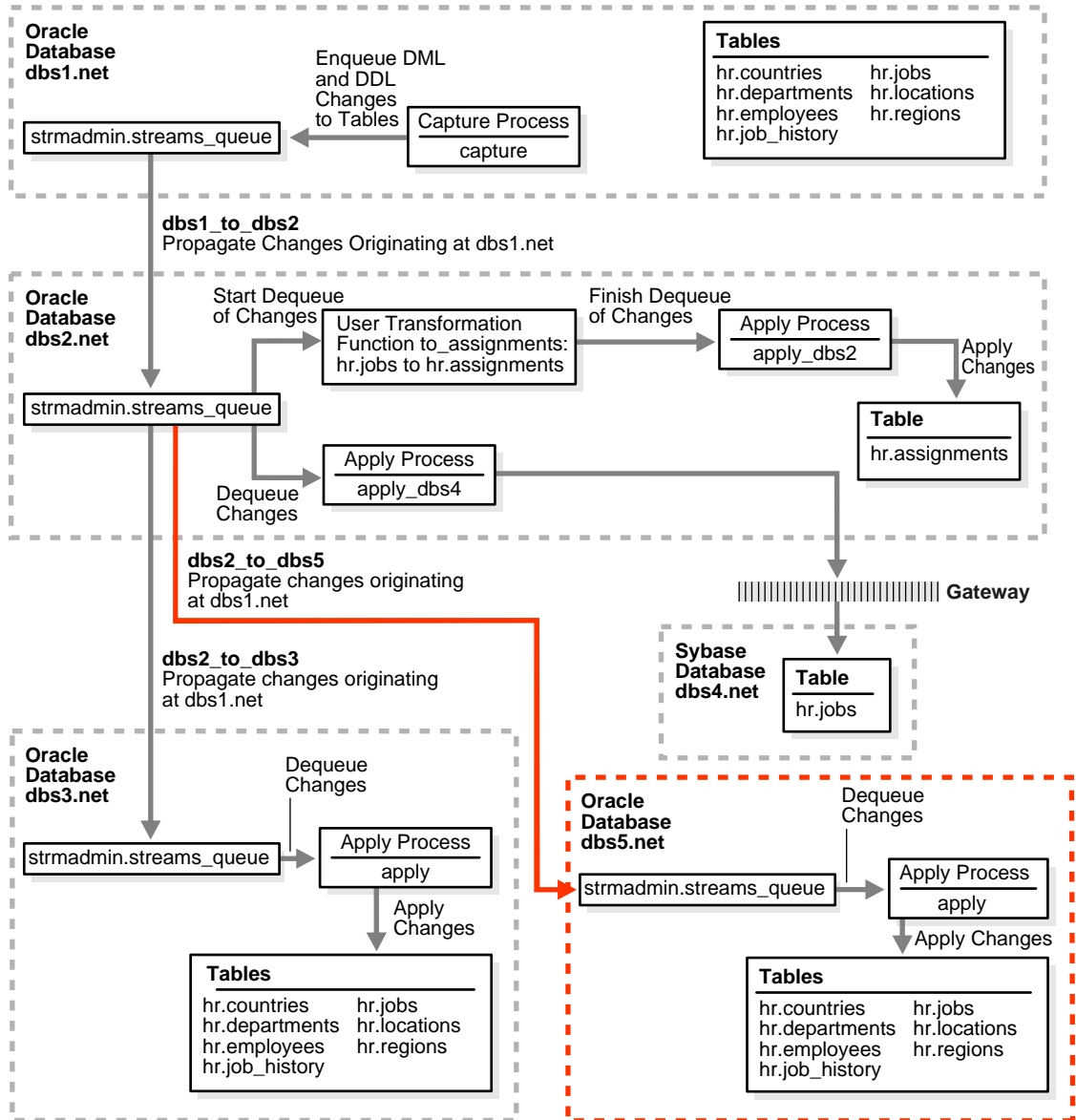
You should see `ST_MAN` for the value of the `job_id`.

Add a Database to an Existing Streams Replication Environment

This example extends the Streams environment configured in the previous sections by adding an additional database to the existing configuration. In this example, an existing Oracle database named `db5.net` is added to receive changes to the entire `hr` schema from the queue at `db2.net`.

[Figure 22-3](#) provides an overview of the environment with the added database.

Figure 22-3 Adding the db5.net Oracle Database to the Environment



To complete this example, you must meet the following prerequisites:

- The `db5.net` database must exist.
- The `db2.net` and `db5.net` databases must be able to communicate with each other through Oracle Net.
- You must have completed the tasks in the previous examples in this chapter.
- The "Prerequisites" on page 22-5 must be met if you want the entire Streams environment to work properly.
- This example creates a new user to function as the Streams administrator (`stradmin`) at each database and prompts you for the tablespace you want to use for this user's data. Before you start this example, either create a new tablespace or identify an existing tablespace for the Streams administrator to use at each database. The Streams administrator should not use the `SYSTEM` tablespace.

Complete the following steps to add `db5.net` to the Streams environment.

1. [Show Output and Spool Results](#)
2. [Drop All of the Tables in the hr Schema at db5.net](#)
3. [Set Up Users at db5.net](#)
4. [Create the Streams Queue at db5.net](#)
5. [Configure the Apply Process at db5.net](#)
6. [Specify hr as the Apply User for the Apply Process at db5.net](#)
7. [Grant the hr User Execute Privilege on the Apply Process Rule Set](#)
8. [Create the Database Link Between db2.net and db5.net](#)
9. [Configure Propagation Between db2.net and db5.net](#)
10. [Prepare the hr Schema for Instantiation at db1.net](#)
11. [Instantiate the db1.net Tables at db5.net](#)
12. [Drop the Supplemental Log Groups at db5.net](#)
13. [Start the Apply Process at db5.net](#)
14. [Check the Spool Results](#)

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the next "END OF SCRIPT" line on page 22-89 into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to all of the databases in the environment.

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

Step 1 Show Output and Spool Results

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

```
*/  
  
SET ECHO ON  
SPOOL streams_adddb.out  
  
/*
```

Step 2 Drop All of the Tables in the hr Schema at dbs5.net

This example illustrates instantiating the tables in the `hr` schema by exporting them from `dbs1.net` and importing them into `dbs5.net`. You must delete these tables at `dbs5.net` for the instantiation portion of this example to work properly.

Connect as `hr` at `dbs5.net`.

```
*/  
  
CONNECT hr/hr@dbs5.net  
  
/*
```

Drop all tables in the `hr` schema in the `dbs5.net` database.

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

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

```
/*
```

Step 3 Set Up Users at dbs5.net

Connect to `dbs5.net` as `SYS` user.

```
*/  
  
CONNECT SYS/CHANGE_ON_INSTALL@dbs5.net AS SYSDBA
```

```
/*
```

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

Note:

- To ensure security, use a password other than `strmadminpw` for the Streams administrator.
 - The `SELECT_CATALOG_ROLE` is not required for the Streams administrator. It is granted in this example so that the Streams administrator can monitor the environment easily.
 - If you plan to use the Streams tool in Oracle Enterprise Manager, then grant the Streams administrator `SELECT ANY DICTIONARY` privilege, in addition to the privileges shown in this step.
 - The `ACCEPT` command must appear on a single line in the script.
-
-

See Also: ["Configuring a Streams Administrator"](#) on page 11-2

```
*/

GRANT CONNECT, RESOURCE, SELECT_CATALOG_ROLE
  TO strmadmin IDENTIFIED BY strmadminpw;

ACCEPT streams_tbs PROMPT 'Enter Streams administrator tablespace on dbs5.net: '

ALTER USER strmadmin DEFAULT TABLESPACE &streams_tbs
  QUOTA UNLIMITED ON &streams_tbs;

GRANT EXECUTE ON DBMS_APPLY_ADM      TO strmadmin;
GRANT EXECUTE ON DBMS_AQADM          TO strmadmin;
GRANT EXECUTE ON DBMS_STREAMS_ADM    TO strmadmin;

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

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

/*
```

Step 4 Create the Streams Queue at dbs5.net

Connect as the Streams administrator at the database you are adding. In this example, that database is `dbs5.net`.

```
*/  
  
CONNECT stradmin/stradminpw@dbs5.net  
  
/*
```

Run the `SET_UP_QUEUE` procedure to create a queue named `streams_queue` at `dbs5.net`. This queue will function as the Streams queue by holding the changes that will be applied at this database.

Running the `SET_UP_QUEUE` procedure performs the following actions:

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

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

Step 5 Configure the Apply Process at dbs5.net

While still connected as the Streams administrator at `dbs5.net`, configure the apply process to apply changes to the `hr` schema.

```
*/  
  
BEGIN  
  DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(  
    schema_name      => 'hr',  
    streams_type     => 'apply',  
    streams_name     => 'apply',  
    queue_name       => 'strmadmin.streams_queue',  
    include_dml      => true,  
    include_ddl      => true,  
    source_database  => 'dbs1.net');  
END;  
/  
  
/*
```

Step 6 Specify hr as the Apply User for the Apply Process at dbs5.net

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

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

See Also: ["Configuring a Streams Administrator"](#) on page 11-2

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

Step 7 Grant the hr User Execute Privilege on the Apply Process Rule Set

Because the `hr` user was specified as the apply user in the previous step, the `hr` user requires execute privilege on the rule set used by the apply process

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

Step 8 Create the Database Link Between `db2.net` and `db5.net`

Connect to `db2.net` as the `strmadmin` user.

```
*/  
  
CONNECT strmadmin/strmadminpw@db2.net  
  
/*
```

Create the database links to the databases where changes are propagated. In this example, database `db5.net` propagates changes to `db1.net`.

```
*/
CREATE DATABASE LINK db5.net CONNECT TO strmadmin IDENTIFIED BY strmadminpw
  USING 'db5.net';
/*
```

Step 9 Configure Propagation Between `db2.net` and `db5.net`

While still connected as the Streams administrator at `db2.net`, Configure and schedule propagation from the queue at `db2.net` to the queue at `db5.net`. Remember, changes to the `hr` schema originated at `db1.net`.

```
*/
BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES(
    schema_name          => 'hr',
    streams_name         => 'db2_to_db5',
    source_queue_name    => 'strmadmin.streams_queue',
    destination_queue_name => 'strmadmin.streams_queue@db5.net',
    include_dml         => true,
    include_ddl         => true,
    source_database      => 'db1.net');
END;
/
/*
```

Step 10 Prepare the `hr` Schema for Instantiation at `db1.net`

Connect to `db1.net` as the `strmadmin` user.

```
*/
CONNECT strmadmin/strmadminpw@db1.net
/*
```

Prepare the `hr` schema for instantiation. These tables in this schema will be instantiated at `db5.net`. This preparation is necessary so that the Streams data dictionary for the relevant propagations and the apply process at `db5.net` contain information about the `hr` schema and the objects in the schema.

See Also:

- ["Preparing Database Objects for Instantiation at a Source Database"](#) on page 12-11
- ["Streams Data Dictionary for Propagations"](#) on page 3-25
- ["Streams Data Dictionary for an Apply Process"](#) on page 4-33

```
*/  
  
BEGIN  
  DBMS_CAPTURE_ADM.PREPARE_SCHEMA_INSTANTIATION(  
    schema_name => 'hr');  
END;  
/  
  
/*
```

Step 11 Instantiate the dbs1.net Tables at dbs5.net

Open a different window and export the schema at `dbs1.net` that will be instantiated at `dbs5.net`. Make sure you set the `OBJECT_CONSISTENT` export parameter to `y` when you run the export command. Also, make sure no DDL changes are made to the objects being exported during the export.

The following is an example export command:

```
exp hr/hr FILE=hr_schema.dmp OWNER=hr OBJECT_CONSISTENT=y
```

See Also: *Oracle9i Database Utilities* for information about performing an export

```
*/  
  
PAUSE Press <RETURN> to continue when the export is complete in the other window  
that you opened.  
  
/*
```

Transfer the export dump file `hr_schema.dmp` to the destination database. In this example, the destination database is `dbs5.net`.

You can use binary FTP or some other method to transfer the export dump file to the destination database. You may need to open a different window to transfer the file.

```
*/
```

PAUSE Press <RETURN> to continue after transferring the dump file.

```
/*
```

In a different window, connect to the computer that runs the `dbs5.net` database and import the export dump file `hr_schema.dmp` to instantiate the tables in the `dbs5.net` database. You can use telnet or remote login to connect to the computer that runs `dbs5.net`.

When you run the import command, make sure you set the `STREAMS_INSTANTIATION` import parameter to `y`. This parameter ensures that the import records export SCN information for each object imported.

The following is an example import command:

```
imp hr/hr FILE=hr_schema.dmp FROMUSER=hr IGNORE=y COMMIT=y LOG=import.log  
STREAMS_INSTANTIATION=y
```

See Also: *Oracle9i Database Utilities* for information about performing an import

```
*/
```

PAUSE Press <RETURN> to continue after the import is complete at `dbs5.net`.

```
/*
```

Step 12 Drop the Supplemental Log Groups at `dbs5.net`

When you instantiated the `hr` schema at `dbs5.net`, the supplemental log groups from `dbs1.net` were retained. These log groups are not needed at `dbs5.net` because no capture process captures changes to the tables in the `hr` schema at `dbs5.net`. You can remove the log groups to avoid extraneous information in the redo log at `dbs5.net`.

Connect to `db55.net` as the `hr` user.

```
*/  
  
CONNECT hr/hr@db55.net
```

```
/*
```

Drop the supplemental log groups at `db55.net`.

```
*/  
  
ALTER TABLE hr.countries DROP SUPPLEMENTAL LOG GROUP log_group_countries_pk;  
  
ALTER TABLE hr.departments DROP SUPPLEMENTAL LOG GROUP log_group_departments_pk;  
  
ALTER TABLE hr.employees DROP SUPPLEMENTAL LOG GROUP log_group_employees_pk;  
  
ALTER TABLE hr.jobs DROP SUPPLEMENTAL LOG GROUP log_group_jobs_pk;  
  
ALTER TABLE hr.job_history DROP SUPPLEMENTAL LOG GROUP log_group_job_history_pk;  
  
ALTER TABLE hr.locations DROP SUPPLEMENTAL LOG GROUP log_group_locations_pk;  
  
ALTER TABLE hr.regions DROP SUPPLEMENTAL LOG GROUP log_group_regions_pk;
```

```
/*
```

Step 13 Start the Apply Process at `db55.net`

Connect as the Streams administrator at `db55.net`.

```
*/  
  
CONNECT strmadmin/strmadminpw@db55.net
```

```
/*
```


Set the `disable_on_error` parameter to `n` so that the apply process will not be disabled if it encounters an error, and start apply process at `dbs5.net`.

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

Step 14 Check the Spool Results

Check the `streams_adddb.out` spool file to ensure that all actions finished successfully after this script is completed.

```
*/  
  
SET ECHO OFF  
SPOOL OFF  
  
/***** END OF SCRIPT *****/
```

Make a DML Change to the hr.departments Table

After completing the examples described in the ["Add a Database to an Existing Streams Replication Environment"](#) section on page 22-76, you can make DML and DDL changes to the tables in the hr schema at the dbs1.net database. These changes will be replicated to dbs5.net. You can check these tables at dbs5.net to see that the changes have been replicated.

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

Step 1 Make a DML Change to the hr.departments Table

Make the following change:

```
CONNECT hr/hr@dbs1.net

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

Step 2 Query the hr.departments Table at dbs5.net

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

```
CONNECT hr/hr@dbs5.net

SELECT location_id FROM hr.departments WHERE department_id=270;
```

You should see 2400 for the value of the location_id.

Multiple Source Replication Example

This chapter illustrates an example of a multiple source replication environment that can be constructed using Streams.

This chapter contains these topics:

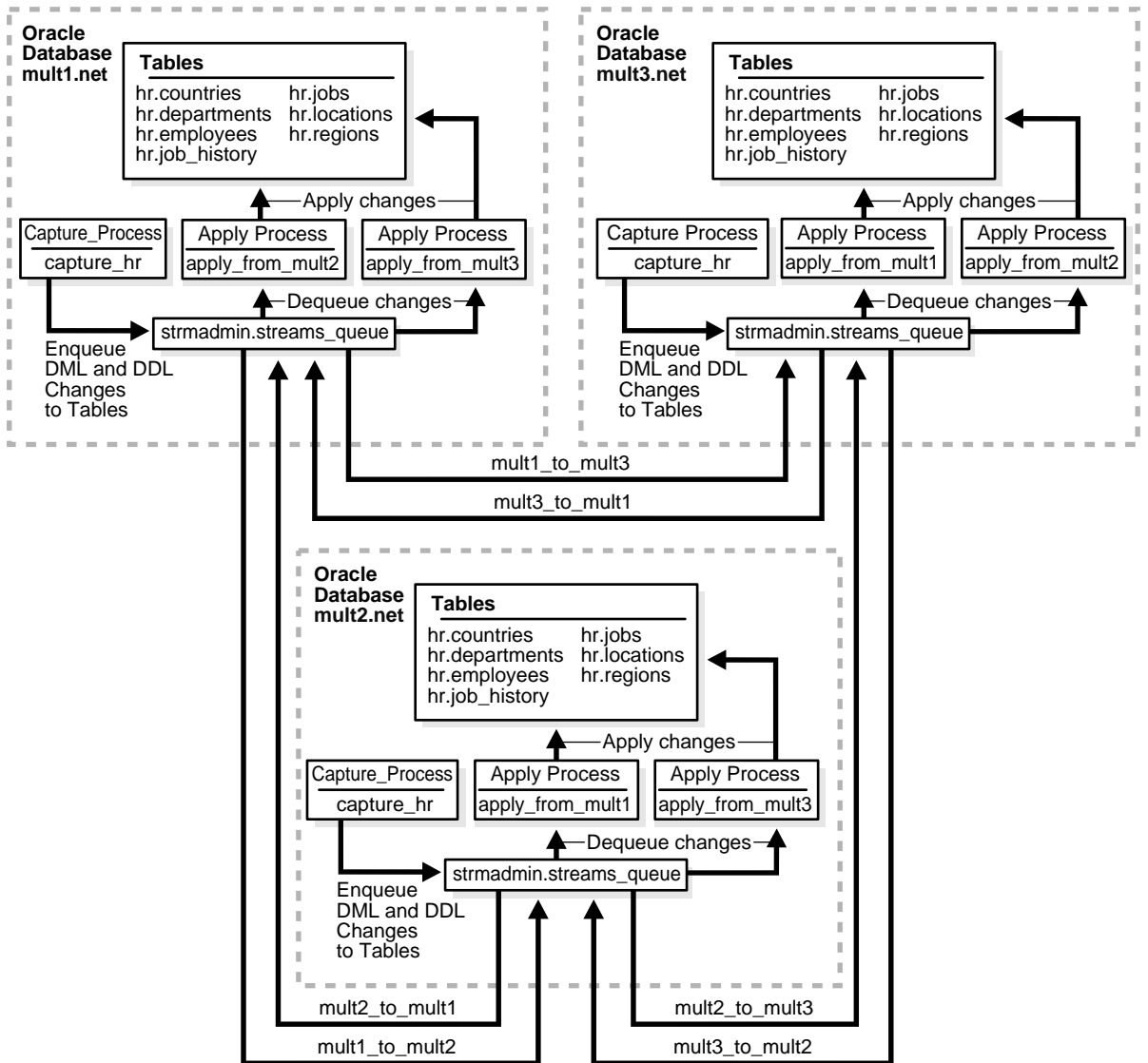
- [Overview of the Multiple Source Databases Example](#)
- [Prerequisites](#)
- [Set Up Users and Create Queues and Database Links](#)
- [Example Script for Sharing Data from Multiple Databases](#)
- [Make DML and DDL Changes to Tables in the hr Schema](#)

Overview of the Multiple Source Databases Example

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

[Figure 23-1](#) provides an overview of the environment.

Figure 23–1 Example Environment That Shares Data from Multiple Databases



As illustrated in [Figure 23-1](#), all of the databases will contain the `hr` schema when the example is complete. However, at the beginning of the example, the `hr` schema exists only at `mult1.net`. During the example, you instantiate the `hr` schema at `mult2.net` and `mult3.net`.

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

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

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

See Also: *Oracle9i Streams* for more information about tags

Prerequisites

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

- Set the following initialization parameters to the values indicated at each database in the Streams environment:
 - `AQ_TM_PROCESSES`: This parameter establishes queue monitor processes. Values from 1 to 10 specify the number of queue monitor processes created to monitor the messages. If `AQ_TM_PROCESSES` is not specified or is set to 0, then the queue monitor processes are not created. In this example, `AQ_TM_PROCESSES` should be set to at least 1.

Setting the parameter to 1 or more starts the specified number of queue monitor processes. These queue monitor processes are responsible for managing time-based operations of messages such as delay and expiration, cleaning up retained messages after the specified retention time, and cleaning up consumed messages if the retention time is 0.
 - `GLOBAL_NAMES`: This parameter must be set to `true`. Make sure the global names of the databases are `mult1.net`, `mult2.net`, and `mult3.net`.
 - `JOB_QUEUE_PROCESSES`: This parameter must be set to at least 2 because each database propagates events. It should be set to the same value as the maximum number of jobs that can run simultaneously plus one.
 - `COMPATIBLE`: This parameter must be set to 9.2.0 or higher.
 - `LOG_PARALLELISM`: This parameter must be set to 1 because each database that captures events.

Attention: You may need to modify other initialization parameter settings for this example to run properly.

See Also: ["Setting Initialization Parameters Relevant to Streams"](#) on page 11-4 for information about other initialization parameters that are important in a Streams environment

- Any database producing changes that will be captured must be running in ARCHIVELOG mode. In this example, all databases are capturing changes, and so all databases must be running in ARCHIVELOG mode.

See Also: *Oracle9i Database Administrator's Guide* for information about running a database in ARCHIVELOG mode

- Configure your network and Oracle Net so that all three databases can communicate with each other.

See Also: *Oracle9i Net Services Administrator's Guide*

Set Up Users and Create Queues and Database Links

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

Complete the following steps to set up the users and to create the `streams_queue` at all of the databases.

1. [Show Output and Spool Results](#)
2. [Alter the hr.countries Table at mult1.net](#)
3. [Create an Alternate Tablespace for the LogMiner Tables at mult1.net](#)
4. [Set Up Users at mult1.net](#)
5. [Create the Streams Queue at mult1.net](#)
6. [Create the Database Links at mult1.net](#)
7. [Prepare the Tables at mult1.net for Latest Time Conflict Resolution](#)
8. [Create an Alternate Tablespace for the LogMiner Tables at mult2.net](#)
9. [Set Up Users at mult2.net](#)
10. [Create the Streams Queue at mult2.net](#)
11. [Create the Database Links at mult2.net](#)
12. [Drop All of the Tables in the hr Schema at mult2.net](#)
13. [Create an Alternate Tablespace for the LogMiner Tables at mult3.net](#)
14. [Set Up Users at mult3.net](#)

15. [Create the Streams Queue at mult3.net](#)
16. [Create the Database Links at mult3.net](#)
17. [Drop All of the Tables in the hr Schema at mult3.net](#)
18. [Check the Spool Results](#)

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the next "END OF SCRIPT" line on page 23-24 into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to all of the databases in the environment.

```

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

```

Step 1 Show Output and Spool Results

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

```

*/

SET ECHO ON
SPOOL streams_setup_mult.out

/*

```

Step 2 Alter the hr.countries Table at mult1.net

Connect to `mult1.net` as the `hr` user.

```

*/

CONNECT hr/hr@mult1.net

/*

```

Convert the `hr.countries` table from an index-organized table to a regular table. Currently, the capture process cannot capture changes to index-organized tables.

```
*/

ALTER TABLE countries RENAME TO countries_orig;

CREATE TABLE hr.countries(
  country_id      CHAR(2) CONSTRAINT country_id_nn_noiot NOT NULL,
  country_name    VARCHAR2(40),
  region_id       NUMBER,
  CONSTRAINT      country_c_id_pk_noiot PRIMARY KEY (country_id));

ALTER TABLE hr.countries
ADD (CONSTRAINT countr_reg_fk_noiot
      FOREIGN KEY (region_id)
      REFERENCES regions(region_id));

INSERT INTO COUNTRIES (SELECT * FROM hr.countries_orig);

DROP TABLE hr.countries_orig CASCADE CONSTRAINTS;

ALTER TABLE locations
  ADD (CONSTRAINT loc_c_id_fk
        FOREIGN KEY (country_id)
        REFERENCES countries(country_id));

/*
```

Step 3 Create an Alternate Tablespace for the LogMiner Tables at mult1.net

By default, the LogMiner tables are in the `SYSTEM` tablespace, but the `SYSTEM` tablespace may not have enough space for these tables once a capture process starts to capture changes. Therefore, you must create an alternate tablespace for the LogMiner tables.

See Also: ["Alternate Tablespace for LogMiner Tables"](#) on page 2-20

Connect to mult1.net as SYS user.

```
*/
CONNECT SYS/CHANGE_ON_INSTALL@mult1.net AS SYSDBA
/*
```

Create an alternate tablespace for the LogMiner tables.

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

```
*/
ACCEPT tspace_name DEFAULT 'logmrts' PROMPT 'Enter the name of the tablespace
(for example, logmrts): '
ACCEPT db_file_directory DEFAULT '' PROMPT 'Enter the complete path to the
datafile directory (for example, /usr/oracle/dbs): '
ACCEPT db_file_name DEFAULT 'logmrts.dbf' PROMPT 'Enter the name of the
datafile (for example, logmrts.dbf): '
CREATE TABLESPACE &tspace_name DATAFILE '&db_file_directory/&db_file_name'
    SIZE 25 M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;
EXECUTE DBMS_LOGMNR_D.SET_TABLESPACE('&tspace_name');
/*
```

Step 4 Set Up Users at mult1.net

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

Note:

- To ensure security, use a password other than `strmadminpw` for the Streams administrator.
 - The `SELECT_CATALOG_ROLE` is not required for the Streams administrator. It is granted in this example so that the Streams administrator can monitor the environment easily.
 - If you plan to use the Streams tool in Oracle Enterprise Manager, then grant the Streams administrator `SELECT ANY DICTIONARY` privilege, in addition to the privileges shown in this step.
 - The `ACCEPT` command must appear on a single line in the script.
-
-

See Also: ["Configuring a Streams Administrator"](#) on page 11-2

```
*/

GRANT CONNECT, RESOURCE, SELECT_CATALOG_ROLE
  TO strmadmin IDENTIFIED BY strmadminpw;

ACCEPT streams_tbs PROMPT 'Enter Streams administrator tablespace on mult1.net:
'

ALTER USER strmadmin DEFAULT TABLESPACE &streams_tbs
  QUOTA UNLIMITED ON &streams_tbs;

GRANT EXECUTE ON DBMS_APPLY_ADM          TO strmadmin;
GRANT EXECUTE ON DBMS_AQADM              TO strmadmin;
GRANT EXECUTE ON DBMS_CAPTURE_ADM       TO strmadmin;
GRANT EXECUTE ON DBMS_PROPAGATION_ADM   TO strmadmin;
GRANT EXECUTE ON DBMS_STREAMS_ADM       TO strmadmin;

BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege => DBMS_RULE_ADM.CREATE_RULE_SET_OBJ,
    grantee   => 'strmadmin',
    grant_option => FALSE);
END;
/
```

```

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

```

```
/*
```

Step 5 Create the Streams Queue at mult1.net

Connect as the Streams administrator at mult1.net.

```
*/
```

```
CONNECT stradmin/stradminpw@mult1.net
```

```
/*
```

Run the `SET_UP_QUEUE` procedure to create a queue named `streams_queue` at `mult1.net`. This queue will function as the Streams queue by holding the changes that will be applied at this database and the changes that will be propagated to other databases.

Running the `SET_UP_QUEUE` procedure performs the following actions:

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

```
*/
```

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

```
/*
```

Step 6 Create the Database Links at mult1.net

Create database links from the current database to the other databases in the environment.

```
*/

CREATE DATABASE LINK mult2.net CONNECT TO strmadmin
  IDENTIFIED BY strmadminpw USING 'mult2.net';

CREATE DATABASE LINK mult3.net CONNECT TO strmadmin
  IDENTIFIED BY strmadminpw USING 'mult3.net';

/*
```

Step 7 Prepare the Tables at mult1.net for Latest Time Conflict Resolution

This example will configure the tables in the hr schema for conflict resolution based on the latest time for a transaction.

Connect to mult1.net as the hr user.

```
*/

CONNECT hr/hr@mult1.net

/*
```

Add a time column to each table in the hr schema.

```
*/

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

/*
```

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

```
*/

CREATE OR REPLACE TRIGGER hr.insert_time_countries
BEFORE
  INSERT OR UPDATE ON hr.countries FOR EACH ROW
BEGIN
  -- Consider time synchronization problems. The previous update to this
  -- row may have originated from a site with a clock time ahead of the
  -- local clock time.
  IF :OLD.TIME IS NULL OR :OLD.TIME < SYSTIMESTAMP THEN
    :NEW.TIME := SYSTIMESTAMP;
  ELSE
    :NEW.TIME := :OLD.TIME + 1 / 86400;
  END IF;
END;
/

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

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

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

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

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



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

/*
```

Step 8 Create an Alternate Tablespace for the LogMiner Tables at mult2.net

By default, the LogMiner tables are in the SYSTEM tablespace, but the SYSTEM tablespace may not have enough space for these tables once a capture process starts to capture changes. Therefore, you must create an alternate tablespace for the LogMiner tables.

See Also: ["Alternate Tablespace for LogMiner Tables"](#) on page 2-20

Connect to mult2.net as SYS user.

```
*/

CONNECT SYS/CHANGE_ON_INSTALL@mult2.net AS SYSDBA

/*
```

Create an alternate tablespace for the LogMiner tables.

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

```
*/

ACCEPT tspace_name DEFAULT 'logmrts' PROMPT 'Enter the name of the tablespace
(for example, logmrts): '

ACCEPT db_file_directory DEFAULT '' PROMPT 'Enter the complete path to the
datafile directory (for example, /usr/oracle/dbs): '

ACCEPT db_file_name DEFAULT 'logmrts.dbf' PROMPT 'Enter the name of the
datafile (for example, logmrts.dbf): '

CREATE TABLESPACE &tspace_name DATAFILE '&db_file_directory/&db_file_name'
SIZE 25 M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;

EXECUTE DBMS_LOGMNR_D.SET_TABLESPACE('&tspace_name');

/*
```

Step 9 Set Up Users at mult2.net

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

Note:

- To ensure security, use a password other than `strmadminpw` for the Streams administrator.
 - The `SELECT_CATALOG_ROLE` is not required for the Streams administrator. It is granted in this example so that the Streams administrator can monitor the environment easily.
 - If you plan to use the Streams tool in Oracle Enterprise Manager, then grant the Streams administrator `SELECT ANY DICTIONARY` privilege, in addition to the privileges shown in this step.
 - The `ACCEPT` command must appear on a single line in the script.
-
-

See Also: ["Configuring a Streams Administrator"](#) on page 11-2

```

*/

GRANT CONNECT, RESOURCE, SELECT_CATALOG_ROLE
  TO strmadmin IDENTIFIED BY strmadminpw;

ACCEPT streams_tbs PROMPT 'Enter Streams administrator tablespace on mult2.net:
'

ALTER USER strmadmin DEFAULT TABLESPACE &streams_tbs
  QUOTA UNLIMITED ON &streams_tbs;

GRANT EXECUTE ON DBMS_APPLY_ADM      TO strmadmin;
GRANT EXECUTE ON DBMS_AQADM          TO strmadmin;
GRANT EXECUTE ON DBMS_CAPTURE_ADM   TO strmadmin;
GRANT EXECUTE ON DBMS_FLASHBACK     TO strmadmin;
GRANT EXECUTE ON DBMS_PROPAGATION_ADM TO strmadmin;
GRANT EXECUTE ON DBMS_STREAMS_ADM    TO strmadmin;

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

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

/*

```

Step 10 Create the Streams Queue at mult2.net

Connect as the Streams administrator at mult2.net.

```

*/

CONNECT strmadmin/strmadminpw@mult2.net

/*

```

Run the `SET_UP_QUEUE` procedure to create a queue named `streams_queue` at `mult2.net`. This queue will function as the Streams queue by holding the changes that will be applied at this database and the changes that will be propagated to other databases.

Running the `SET_UP_QUEUE` procedure performs the following actions:

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

```
*/  
  
EXEC DEMS_STREAMS_ADM.SET_UP_QUEUE();  
  
/*
```

Step 11 Create the Database Links at `mult2.net`

Create database links from the current database to the other databases in the environment.

```
*/  
  
CREATE DATABASE LINK mult1.net CONNECT TO strmadmin  
  IDENTIFIED BY strmadminpw USING 'mult1.net';  
  
CREATE DATABASE LINK mult3.net CONNECT TO strmadmin  
  IDENTIFIED BY strmadminpw USING 'mult3.net';  
  
/*
```

Step 12 Drop All of the Tables in the `hr` Schema at `mult2.net`

This example illustrates instantiating the tables in the `hr` schema at `mult2.net` by exporting these tables from `mult1.net` and importing them into `mult2.net`. You must drop the tables in the `hr` schema at `mult2.net` for the instantiation portion of this example to work properly.

Attention: If you complete the following steps and drop the tables in the `hr` schema at `mult2.net`, then you should complete the remaining steps of this example to reinstantiate the `hr` schema. If the `hr` schema does not exist in an Oracle database, then some examples in the Oracle documentation set may fail.

Connect as `hr` at `mult2.net`.

```
*/  
  
CONNECT hr/hr@mult2.net  
  
/*
```

Drop all tables in the `hr` schema in the `mult2.net` database.

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

Step 13 Create an Alternate Tablespace for the LogMiner Tables at `mult3.net`

By default, the LogMiner tables are in the `SYSTEM` tablespace, but the `SYSTEM` tablespace may not have enough space for these tables once a capture process starts to capture changes. Therefore, you must create an alternate tablespace for the LogMiner tables.

See Also: ["Alternate Tablespace for LogMiner Tables"](#) on page 2-20

Connect to mult3.net as SYS user.

```
*/  
  
CONNECT SYS/CHANGE_ON_INSTALL@mult3.net AS SYSDBA  
  
/*
```

Create an alternate tablespace for the LogMiner tables.

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

```
*/  
  
ACCEPT tspace_name DEFAULT 'logmrts' PROMPT 'Enter the name of the tablespace  
(for example, logmrts): '  
  
ACCEPT db_file_directory DEFAULT '' PROMPT 'Enter the complete path to the  
datafile directory (for example, /usr/oracle/dbs): '  
  
ACCEPT db_file_name DEFAULT 'logmrts.dbf' PROMPT 'Enter the name of the  
datafile (for example, logmrts.dbf): '  
  
CREATE TABLESPACE &tspace_name DATAFILE '&db_file_directory/&db_file_name'  
    SIZE 25 M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;  
  
EXECUTE DBMS_LOGMNR_D.SET_TABLESPACE('&tspace_name');  
  
/*
```

Step 14 Set Up Users at mult3.net

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

Note:

- To ensure security, use a password other than `strmadminpw` for the Streams administrator.
- The `SELECT_CATALOG_ROLE` is not required for the Streams administrator. It is granted in this example so that the Streams administrator can monitor the environment easily.
- If you plan to use the Streams tool in Oracle Enterprise Manager, then grant the Streams administrator `SELECT ANY DICTIONARY` privilege, in addition to the privileges shown in this step.
- The `ACCEPT` command must appear on a single line in the script.

See Also: ["Configuring a Streams Administrator"](#) on page 11-2

```

*/

GRANT CONNECT, RESOURCE, SELECT_CATALOG_ROLE
  TO strmadmin IDENTIFIED BY strmadminpw;

ACCEPT streams_tbs PROMPT 'Enter Streams administrator tablespace on mult3.net:
'

ALTER USER strmadmin DEFAULT TABLESPACE &streams_tbs
  QUOTA UNLIMITED ON &streams_tbs;

GRANT EXECUTE ON DBMS_APPLY_ADM          TO strmadmin;
GRANT EXECUTE ON DBMS_AQADM              TO strmadmin;
GRANT EXECUTE ON DBMS_CAPTURE_ADM       TO strmadmin;
GRANT EXECUTE ON DBMS_FLASHBACK          TO strmadmin;
GRANT EXECUTE ON DBMS_PROPAGATION_ADM   TO strmadmin;
GRANT EXECUTE ON DBMS_STREAMS_ADM       TO strmadmin;

```

```
BEGIN
  DBMS_RULE_ADM.GRANT_SYSTEM_PRIVILEGE(
    privilege    => DBMS_RULE_ADM.CREATE_RULE_SET_OBJ,
    grantee      => 'strmadmin',
    grant_option => FALSE);
END;
/

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

/*
```

Step 15 Create the Streams Queue at mult3.net

Connect as the Streams administrator at mult3.net.

```
*/

CONNECT strmadmin/strmadminpw@mult3.net

/*
```

Run the `SET_UP_QUEUE` procedure to create a queue named `streams_queue` at `mult3.net`. This queue will function as the Streams queue by holding the changes that will be applied at this database and the changes that will be propagated to other databases.

Running the `SET_UP_QUEUE` procedure performs the following actions:

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


```
*/
EXEC DEMS_STREAMS_ADM.SET_UP_QUEUE();
/*
```

Step 16 Create the Database Links at mult3.net

Create database links from the current database to the other databases in the environment.

```
*/
CREATE DATABASE LINK mult1.net CONNECT TO strmadmin
  IDENTIFIED BY strmadminpw USING 'mult1.net';

CREATE DATABASE LINK mult2.net CONNECT TO strmadmin
  IDENTIFIED BY strmadminpw USING 'mult2.net';
/*
```

Step 17 Drop All of the Tables in the hr Schema at mult3.net

This example illustrates instantiating the tables in the `hr` schema at `mult3.net` by exporting these tables from `mult1.net` and importing them into `mult3.net`. You must drop the tables in the `hr` schema at `mult3.net` for the instantiation portion of this example to work properly.

Attention: If you complete the following steps and drop the tables in the `hr` schema at `mult3.net`, then you should complete the remaining steps of this example to reinstantiate the `hr` schema. If the `hr` schema does not exist in an Oracle database, then some examples in the Oracle documentation set may fail.

Connect as `hr` at `mult3.net`.

```
*/
CONNECT hr/hr@mult3.net
/*
```

Drop all tables in the hr schema in the mult3.net database.

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

Step 18 Check the Spool Results

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

```
*/  
  
SET ECHO OFF  
SPOOL OFF  
  
/***** END OF SCRIPT *****/
```

Example Script for Sharing Data from Multiple Databases

Complete the following steps to configure a Streams environment that shares information from multiple databases.

1. Show Output and Spool Results
2. Specify Supplemental Logging at mult1.net
3. Create the Capture Process at mult1.net
4. Create One Apply Process at mult1.net for Each Source Database
5. Specify hr as the Apply User for Each Apply Process at mult1.net
6. Grant the hr User Execute Privilege on the Apply Process Rule Set
7. Configure Latest Time Conflict Resolution at mult1.net
8. Configure Propagation at mult1.net
9. Create the Capture Process at mult2.net.
10. Set the Instantiation SCN for mult2.net at the Other Databases
11. Create One Apply Process at mult2.net for Each Source Database
12. Specify hr as the Apply User for Each Apply Process at mult2.net
13. Grant the hr User Execute Privilege on the Apply Process Rule Set
14. Configure Propagation at mult2.net
15. Create the Capture Process at mult3.net
16. Set the Instantiation SCN for mult3.net at the Other Databases
17. Create One Apply Process at mult3.net for Each Source Database
18. Specify hr as the Apply User for Each Apply Process at mult3.net
19. Grant the hr User Execute Privilege on the Apply Process Rule Set
20. Configure Propagation at mult3.net
21. Instantiate the hr Schema at mult2.net and mult3.net
22. Configure Latest Time Conflict Resolution at mult2.net
23. Start the Apply Processes at mult2.net
24. Configure Latest Time Conflict Resolution at mult3.net
25. Start the Apply Processes at mult3.net

26. [Start the Apply Processes at mult1.net](#)
27. [Start the Capture Process at mult1.net](#)
28. [Start the Capture Process at mult2.net](#)
29. [Start the Capture Process at mult3.net](#)
30. [Check the Spool Results](#)

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the next "END OF SCRIPT" line on page 23-61 into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to all of the databases in the environment.

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

Step 1 Show Output and Spool Results

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

```
*/

SET ECHO ON
SPOOL streams_mult.out

/*
```

Step 2 Specify Supplemental Logging at mult1.net

Connect to mult1.net as SYS user.

```
*/

CONNECT SYS/CHANGE_ON_INSTALL@mult1.net AS SYSDBA

/*
```

Specify an unconditional supplemental log group that includes the primary key for each table and the column list for each table, as specified in "[Configure Latest Time Conflict Resolution at mult1.net](#)" on page 23-32.

Note:

- For convenience, this example includes the primary key column(s) for each table and the columns used for update conflict resolution in a single unconditional log group. You may choose to place the primary key column(s) for each table in an unconditional log group and the columns used for update conflict resolution in a conditional log group.
 - You do not need to specify supplemental logging explicitly at `mult2.net` and `mult3.net` in this example. When you use `export/import` to instantiate the tables in the `hr` schema at these databases later in this example, the supplemental logging specifications at `mult1.net` are retained at `mult2.net` and `mult3.net`.
-

See Also:

- ["Supplemental Logging in a Streams Environment"](#) on page 2-11
- ["Specifying Supplemental Logging at a Source Database"](#) on page 12-9

```
*/
```

```
ALTER TABLE hr.countries ADD SUPPLEMENTAL LOG GROUP log_group_countries  
(country_id, country_name, region_id, time) ALWAYS;
```

```
ALTER TABLE hr.departments ADD SUPPLEMENTAL LOG GROUP log_group_departments  
(department_id, department_name, manager_id, location_id, time) ALWAYS;
```

```
ALTER TABLE hr.employees ADD SUPPLEMENTAL LOG GROUP log_group_employees  
(employee_id, first_name, last_name, email, phone_number, hire_date, job_id,  
salary, commission_pct, manager_id, department_id, time) ALWAYS;
```

```
ALTER TABLE hr.jobs ADD SUPPLEMENTAL LOG GROUP log_group_jobs  
(job_id, job_title, min_salary, max_salary, time) ALWAYS;
```

```
ALTER TABLE hr.job_history ADD SUPPLEMENTAL LOG GROUP log_group_job_history  
(employee_id, start_date, end_date, job_id, department_id, time) ALWAYS;
```

```
ALTER TABLE hr.locations ADD SUPPLEMENTAL LOG GROUP log_group_locations
(location_id, street_address, postal_code, city, state_province,
country_id, time) ALWAYS;
```

```
ALTER TABLE hr.regions ADD SUPPLEMENTAL LOG GROUP log_group_regions
(region_id, region_name, time) ALWAYS;
```

```
/*
```

Step 3 Create the Capture Process at mult1.net

Connect to mult1.net as the strmadmin user.

```
*/
```

```
CONNECT strmadmin/strmadminpw@mult1.net
```

```
/*
```

Create the capture process to capture changes to the entire hr schema at mult1.net. After this step is complete, users can modify tables in the hr schema at mult1.net.

```
*/
```

```
BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
    schema_name => 'hr',
    streams_type => 'capture',
    streams_name => 'capture_hr',
    queue_name => 'strmadmin.streams_queue',
    include_dml => true,
    include_ddl => true);
```

```
END;
```

```
/
```

```
/*
```

Step 4 Create One Apply Process at mult1.net for Each Source Database

Configure mult1.net to apply changes to the hr schema at mult2.net.

```
*/  
  
BEGIN  
  DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(  
    schema_name      => 'hr',  
    streams_type     => 'apply',  
    streams_name     => 'apply_from_mult2',  
    queue_name       => 'strmadmin.streams_queue',  
    include_dml      => true,  
    include_ddl      => true,  
    source_database  => 'mult2.net');  
END;  
/  
  
/*
```

Configure mult1.net to apply changes to the hr schema at mult3.net.

```
*/  
  
BEGIN  
  DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(  
    schema_name      => 'hr',  
    streams_type     => 'apply',  
    streams_name     => 'apply_from_mult3',  
    queue_name       => 'strmadmin.streams_queue',  
    include_dml      => true,  
    include_ddl      => true,  
    source_database  => 'mult3.net');  
END;  
/  
  
/*
```

Step 5 Specify hr as the Apply User for Each Apply Process at mult1.net

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

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

See Also: ["Configuring a Streams Administrator"](#) on page 11-2

```
*/

BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name => 'apply_from_mult2',
    apply_user => 'hr');
END;
/

BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name => 'apply_from_mult3',
    apply_user => 'hr');
END;
/

/*
```


Step 6 Grant the hr User Execute Privilege on the Apply Process Rule Set

Because the `hr` user was specified as the `apply` user in the previous step, the `hr` user requires execute privilege on the rule set used by each apply process

```

*/

DECLARE
  rs_name VARCHAR2(64);  -- Variable to hold rule set name
BEGIN
  SELECT RULE_SET_OWNER || '.' || RULE_SET_NAME
  INTO rs_name
  FROM DBA_APPLY
  WHERE APPLY_NAME='APPLY_FROM_MULT2';
  DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE(
    privilege => SYS.DBMS_RULE_ADM.EXECUTE_ON_RULE_SET,
    object_name => rs_name,
    grantee    => 'hr');
END;
/

DECLARE
  rs_name VARCHAR2(64);  -- Variable to hold rule set name
BEGIN
  SELECT RULE_SET_OWNER || '.' || RULE_SET_NAME
  INTO rs_name
  FROM DBA_APPLY
  WHERE APPLY_NAME='APPLY_FROM_MULT3';
  DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE(
    privilege => SYS.DBMS_RULE_ADM.EXECUTE_ON_RULE_SET,
    object_name => rs_name,
    grantee    => 'hr');
END;
/

/*

```

Step 7 Configure Latest Time Conflict Resolution at mult1.net

Specify an update conflict handler for each table in the `hr` schema. For each table, designate the `time` column as the resolution column for a `MAXIMUM` conflict handler. When an update conflict occurs, such an update conflict handler applies the transaction with the latest (or greater) time and discards the transaction with the earlier (or lesser) time. The column lists for each table do not include the primary key because this example assumes that primary key values are never updated.

```

*/

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

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

```

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

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

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

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

```

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

/*

```

Step 8 Configure Propagation at mult1.net

Configure and schedule propagation of DML and DDL changes in the hr schema from the queue at mult1.net to the queue at mult2.net.

```

*/

BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES(
    schema_name      => 'hr',
    streams_name     => 'mult1_to_mult2',
    source_queue_name => 'strmadmin.streams_queue',
    destination_queue_name => 'strmadmin.streams_queue@mult2.net',
    include_dml      => true,
    include_ddl      => true,
    source_database  => 'mult1.net');
END;
/

/*

```

Configure and schedule propagation of DML and DDL changes in the hr schema from the queue at mult1.net to the queue at mult3.net.

```
*/

BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES(
    schema_name          => 'hr',
    streams_name         => 'mult1_to_mult3',
    source_queue_name    => 'strmadmin.streams_queue',
    destination_queue_name => 'strmadmin.streams_queue@mult3.net',
    include_dml          => true,
    include_ddl          => true,
    source_database      => 'mult1.net');
END;
/

/*
```

Step 9 Create the Capture Process at mult2.net.

Connect to mult2.net as the strmadmin user.

```
*/

CONNECT strmadmin/strmadminpw@mult2.net

/*
```

Create the capture process to capture changes to the entire hr schema at mult2.net.

```
*/

BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
    schema_name  => 'hr',
    streams_type => 'capture',
    streams_name => 'capture_hr',
    queue_name   => 'strmadmin.streams_queue',
    include_dml => true,
    include_ddl => true);
END;
/

/*
```

Step 10 Set the Instantiation SCN for mult2.net at the Other Databases

In this example, the `hr` schema already exists at all of the databases. The tables in the schema exist only at `mult1.net` until they are instantiated at `mult2.net` and `mult3.net` in Step 21. The instantiation is done using an export of the tables from `mult1.net`. These export/import operations set the schema instantiation SCNs for `mult1.net` at `mult2.net` and `mult3.net` automatically.

However, the instantiation SCNs for `mult2.net` and `mult3.net` are not set automatically at the other sites in the environment. This step sets the schema instantiation SCN for `mult2.net` manually at `mult1.net` and `mult3.net`. The current SCN at `mult2.net` is obtained by using the `GET_SYSTEM_CHANGE_NUMBER` function in the `DBMS_FLASHBACK` package at `mult2.net`. This SCN is used at `mult1.net` and `mult3.net` to run the `SET_SCHEMA_INSTANTIATION_SCN` procedure in the `DBMS_APPLY_ADM` package.

The `SET_SCHEMA_INSTANTIATION_SCN` procedure controls which DDL LCRs for a schema are ignored by an apply process and which DDL LCRs for a schema are applied by an apply process. If the commit SCN of a DDL LCR for a database object in a schema from a source database is less than or equal to the instantiation SCN for that database object at some destination database, then the apply process at the destination database disregards the DDL LCR. Otherwise, the apply process applies the DDL LCR.

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

Note:

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

```
*/  
  
DECLARE  
    iscn NUMBER;          -- Variable to hold instantiation SCN value  
BEGIN  
    iscn := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER();  
    DBMS_APPLY_ADM.SET_SCHEMA_INSTANTIATION_SCN@MULT1.NET(  
        source_schema_name => 'hr',  
        source_database_name => 'mult2.net',  
        instantiation_scn => iscn);  
    DBMS_APPLY_ADM.SET_SCHEMA_INSTANTIATION_SCN@MULT3.NET(  
        source_schema_name => 'hr',  
        source_database_name => 'mult2.net',  
        instantiation_scn => iscn);  
END;  
/  
  
/*
```

Step 11 Create One Apply Process at mult2.net for Each Source Database Configure mult2.net to apply changes to the hr schema at mult1.net.

```
*/  
  
BEGIN  
    DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(  
        schema_name => 'hr',  
        streams_type => 'apply',  
        streams_name => 'apply_from_mult1',  
        queue_name => 'strmadmin.streams_queue',  
        include_dml => true,  
        include_ddl => true,  
        source_database => 'mult1.net');  
END;  
/  
  
/*
```


Configure `mult2.net` to apply changes to the `hr` schema at `mult3.net`.

```

*/

BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
    schema_name      => 'hr',
    streams_type     => 'apply',
    streams_name     => 'apply_from_mult3',
    queue_name       => 'strmadmin.streams_queue',
    include_dml      => true,
    include_ddl      => true,
    source_database  => 'mult3.net');
END;
/

/*

```

Step 12 Specify `hr` as the Apply User for Each Apply Process at `mult2.net`

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

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

See Also: ["Configuring a Streams Administrator"](#) on page 11-2

```
*/  
  
BEGIN  
  DBMS_APPLY_ADM.ALTER_APPLY(  
    apply_name => 'apply_from_mult1',  
    apply_user => 'hr');  
END;  
/  
  
BEGIN  
  DBMS_APPLY_ADM.ALTER_APPLY(  
    apply_name => 'apply_from_mult3',  
    apply_user => 'hr');  
END;  
/  
  
/*
```

Step 13 Grant the hr User Execute Privilege on the Apply Process Rule Set

Because the `hr` user was specified as the apply user in the previous step, the `hr` user requires execute privilege on the rule set used by each apply process

```
*/  
  
DECLARE  
  rs_name VARCHAR2(64); -- Variable to hold rule set name  
BEGIN  
  SELECT RULE_SET_OWNER || '.' || RULE_SET_NAME  
    INTO rs_name  
  FROM DBA_APPLY  
  WHERE APPLY_NAME='APPLY_FROM_MULT1';  
  DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE(  
    privilege => SYS.DBMS_RULE_ADM.EXECUTE_ON_RULE_SET,  
    object_name => rs_name,  
    grantee => 'hr');  
END;  
/  
  
/*
```

```

DECLARE
  rs_name VARCHAR2(64);  -- Variable to hold rule set name
BEGIN
  SELECT RULE_SET_OWNER || '.' || RULE_SET_NAME
         INTO rs_name
         FROM DBA_APPLY
         WHERE APPLY_NAME='APPLY_FROM_MULT3';
  DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE(
    privilege => SYS.DBMS_RULE_ADM.EXECUTE_ON_RULE_SET,
    object_name => rs_name,
    grantee    => 'hr');
END;
/

/*

```

Step 14 Configure Propagation at mult2.net

Configure and schedule propagation of DML and DDL changes in the hr schema from the queue at mult2.net to the queue at mult1.net.

```

*/

BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES(
    schema_name          => 'hr',
    streams_name         => 'mult2_to_mult1',
    source_queue_name    => 'strmadmin.streams_queue',
    destination_queue_name => 'strmadmin.streams_queue@mult1.net',
    include_dml          => true,
    include_ddl          => true,
    source_database      => 'mult2.net');
END;
/

/*

```

Configure and schedule propagation of DML and DDL changes in the hr schema from the queue at mult2.net to the queue at mult3.net.

```
*/

BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES(
    schema_name      => 'hr',
    streams_name     => 'mult2_to_mult3',
    source_queue_name => 'strmadmin.streams_queue',
    destination_queue_name => 'strmadmin.streams_queue@mult3.net',
    include_dml      => true,
    include_ddl      => true,
    source_database  => 'mult2.net');
END;
/

/*
```

Step 15 Create the Capture Process at mult3.net

Connect to mult3.net as the strmadmin user.

```
*/

CONNECT strmadmin/strmadminpw@mult3.net

/*
```

Create the capture process to capture changes to the entire hr schema at mult3.net.

```
*/

BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
    schema_name  => 'hr',
    streams_type => 'capture',
    streams_name => 'capture_hr',
    queue_name   => 'strmadmin.streams_queue',
    include_dml  => true,
    include_ddl  => true);
END;
/

/*
```

Step 16 Set the Instantiation SCN for mult3.net at the Other Databases

In this example, the `hr` schema already exists at all of the databases. The tables in the schema exist only at `mult1.net` until they are instantiated at `mult2.net` and `mult3.net` in Step 21. The instantiation is done using an export of the tables from `mult1.net`. These export/import operations set the schema instantiation SCNs for `mult1.net` at `mult2.net` and `mult3.net` automatically.

However, the instantiation SCNs for `mult2.net` and `mult3.net` are not set automatically at the other sites in the environment. This step sets the schema instantiation SCN for `mult3.net` manually at `mult1.net` and `mult2.net`. The current SCN at `mult3.net` is obtained by using the `GET_SYSTEM_CHANGE_NUMBER` function in the `DBMS_FLASHBACK` package at `mult3.net`. This SCN is used at `mult1.net` and `mult2.net` to run the `SET_SCHEMA_INSTANTIATION_SCN` procedure in the `DBMS_APPLY_ADM` package.

The `SET_SCHEMA_INSTANTIATION_SCN` procedure controls which DDL LCRs for a schema are ignored by an apply process and which DDL LCRs for a schema are applied by an apply process. If the commit SCN of a DDL LCR for a database object in a schema from a source database is less than or equal to the instantiation SCN for that database object at some destination database, then the apply process at the destination database disregards the DDL LCR. Otherwise, the apply process applies the DDL LCR.

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

Note:

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

```
*/  
  
DECLARE  
    iscn NUMBER;          -- Variable to hold instantiation SCN value  
BEGIN  
    iscn := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER();  
    DBMS_APPLY_ADM.SET_SCHEMA_INSTANTIATION_SCN@MULT1.NET(  
        source_schema_name => 'hr',  
        source_database_name => 'mult3.net',  
        instantiation_scn => iscn);  
    DBMS_APPLY_ADM.SET_SCHEMA_INSTANTIATION_SCN@MULT2.NET(  
        source_schema_name => 'hr',  
        source_database_name => 'mult3.net',  
        instantiation_scn => iscn);  
END;  
/  
  
/*
```

Step 17 Create One Apply Process at mult3.net for Each Source Database Configure mult3.net to apply changes to the hr schema at mult1.net.

```
*/  
  
BEGIN  
    DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(  
        schema_name => 'hr',  
        streams_type => 'apply',  
        streams_name => 'apply_from_mult1',  
        queue_name => 'strmadmin.streams_queue',  
        include_dml => true,  
        include_ddl => true,  
        source_database => 'mult1.net');  
END;  
/  
  
/*
```

Configure `mult3.net` to apply changes to the `hr` schema at `mult2.net`.

```

*/

BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
    schema_name      => 'hr',
    streams_type     => 'apply',
    streams_name     => 'apply_from_mult2',
    queue_name       => 'strmadmin.streams_queue',
    include_dml      => true,
    include_ddl      => true,
    source_database  => 'mult2.net');
END;
/

/*

```

Step 18 Specify `hr` as the Apply User for Each Apply Process at `mult3.net`

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

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

See Also: ["Configuring a Streams Administrator"](#) on page 11-2

```
*/  
  
BEGIN  
  DBMS_APPLY_ADM.ALTER_APPLY(  
    apply_name => 'apply_from_mult1',  
    apply_user => 'hr');  
END;  
/  
  
BEGIN  
  DBMS_APPLY_ADM.ALTER_APPLY(  
    apply_name => 'apply_from_mult2',  
    apply_user => 'hr');  
END;  
/  
  
/*
```

Step 19 Grant the hr User Execute Privilege on the Apply Process Rule Set

Because the `hr` user was specified as the apply user in the previous step, the `hr` user requires execute privilege on the rule set used by each apply process

```
*/  
  
DECLARE  
  rs_name VARCHAR2(64); -- Variable to hold rule set name  
BEGIN  
  SELECT RULE_SET_OWNER || '.' || RULE_SET_NAME  
    INTO rs_name  
  FROM DBA_APPLY  
  WHERE APPLY_NAME='APPLY_FROM_MULT1';  
  DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE(  
    privilege => SYS.DBMS_RULE_ADM.EXECUTE_ON_RULE_SET,  
    object_name => rs_name,  
    grantee => 'hr');  
END;  
/  
  
/*
```



```

DECLARE
  rs_name VARCHAR2(64);  -- Variable to hold rule set name
BEGIN
  SELECT RULE_SET_OWNER || '.' || RULE_SET_NAME
         INTO rs_name
         FROM DBA_APPLY
         WHERE APPLY_NAME='APPLY_FROM_MULT2';
  DBMS_RULE_ADM.GRANT_OBJECT_PRIVILEGE(
    privilege => SYS.DBMS_RULE_ADM.EXECUTE_ON_RULE_SET,
    object_name => rs_name,
    grantee    => 'hr');
END;
/

/*

```

Step 20 Configure Propagation at mult3.net

Configure and schedule propagation of DML and DDL changes in the hr schema from the queue at mult3.net to the queue at mult1.net.

```

*/

BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES(
    schema_name          => 'hr',
    streams_name         => 'mult3_to_mult1',
    source_queue_name    => 'strmadmin.streams_queue',
    destination_queue_name => 'strmadmin.streams_queue@mult1.net',
    include_dml          => true,
    include_ddl          => true,
    source_database      => 'mult3.net');
END;
/

/*

```

Configure and schedule propagation of DML and DDL changes in the hr schema from the queue at mult3.net to the queue at mult2.net.

```
*/

BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES(
    schema_name      => 'hr',
    streams_name     => 'mult3_to_mult2',
    source_queue_name => 'strmadmin.streams_queue',
    destination_queue_name => 'strmadmin.streams_queue@mult2.net',
    include_dml      => true,
    include_ddl      => true,
    source_database  => 'mult3.net');
END;
/

/*
```

Step 21 Instantiate the hr Schema at mult2.net and mult3.net

Open a different window and export the schema at mult1.net that will be instantiated at mult2.net and mult3.net. Make sure you set the OBJECT_CONSISTENT export parameter to y when you run the export command. Also, make sure no DDL changes are made to the objects being exported during the export.

The following is an example export command:

```
exp hr/hr FILE=hr_schema.dmp OWNER=hr OBJECT_CONSISTENT=y
```

See Also: *Oracle9i Database Utilities* for information about performing an export

```
*/

PAUSE Press <RETURN> to continue when the export is complete in the other window
that you opened.

/*
```

Transfer the export dump file hr_schema.dmp to the destination databases. In this example, the destination databases are mult2.net and mult3.net.

You can use binary FTP or some other method to transfer the export dump file to the destination database. You may need to open a different window to transfer the file.

```
*/
```

PAUSE Press <RETURN> to continue after transferring the dump file to all of the other databases in the environment.

```
/*
```

In a different window, connect to the computer that runs the `mult2.net` database and import the export dump file `hr_schema.dmp` to instantiate the tables in the `mult2.net` database. You can use telnet or remote login to connect to the computer that runs `mult2.net`.

When you run the import command, make sure you set the `STREAMS_INSTANTIATION` import parameter to `y`. This parameter ensures that the import records export SCN information for each object imported.

Also, make sure no changes are made to the tables in the schema being imported at the destination database (`mult2.net`) until the import is complete and the capture process is created.

The following is an example import command:

```
imp hr/hr FILE=hr_schema.dmp FROMUSER=hr IGNORE=y COMMIT=y LOG=import.log  
STREAMS_INSTANTIATION=y
```

See Also: *Oracle9i Database Utilities* for information about performing an import

```
*/
```

PAUSE Press <RETURN> to continue after the import is complete at `mult2.net`.

```
/*
```

In a different window, connect to the computer that runs the `mult3.net` database and import the export dump file `hr_schema.dmp` to instantiate the tables in the `mult3.net` database.

After you connect to `mult3.net`, perform the import in the same way that you did for `mult2.net`.

```
*/
```

PAUSE Press <RETURN> to continue after the import is complete at `mult3.net`.

```
/*
```

Step 22 Configure Latest Time Conflict Resolution at `mult2.net`

Connect to `mult2.net` as the `strmadmin` user.

```
*/
```

```
CONNECT strmadmin/strmadminpw@mult2.net
```

```
/*
```

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

```
*/
```

```
DECLARE
```

```
  cols DBMS_UTILITY.NAME_ARRAY;
```

```
BEGIN
```

```
  cols(1) := 'country_name';
```

```
  cols(2) := 'region_id';
```

```
  cols(3) := 'time';
```

```
  DBMS_APPLY_ADM.SET_UPDATE_CONFLICT_HANDLER(
```

```
    object_name      => 'hr.countries',
```

```
    method_name      => 'MAXIMUM',
```

```
    resolution_column => 'time',
```

```
    column_list      => cols);
```

```
END;
```

```
/
```

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

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

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

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

```

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

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

/*

```

Step 23 Start the Apply Processes at mult2.net

Set the `disable_on_error` parameter to `n` for both apply processes so that they will not be disabled if they encounter an error, and start both of the apply processes at `mult2.net`.

```
*/  
  
BEGIN  
  DBMS_APPLY_ADM.SET_PARAMETER(  
    apply_name => 'apply_from_mult1',  
    parameter  => 'disable_on_error',  
    value      => 'n');  
END;  
/  
  
BEGIN  
  DBMS_APPLY_ADM.START_APPLY(  
    apply_name => 'apply_from_mult1');  
END;  
/  
  
BEGIN  
  DBMS_APPLY_ADM.SET_PARAMETER(  
    apply_name => 'apply_from_mult3',  
    parameter  => 'disable_on_error',  
    value      => 'n');  
END;  
/  
  
BEGIN  
  DBMS_APPLY_ADM.START_APPLY(  
    apply_name => 'apply_from_mult3');  
END;  
/  
  
/*
```

Step 24 Configure Latest Time Conflict Resolution at mult3.net

Connect to mult3.net as the strmadmin user.

```
*/  
  
CONNECT strmadmin/strmadminpw@mult3.net  
  
/*
```


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

```
*/

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

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

Example Script for Sharing Data from Multiple Databases

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

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

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

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

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

/*
```

Step 25 Start the Apply Processes at mult3.net

Set the `disable_on_error` parameter to `n` for both apply processes so that they will not be disabled if they encounter an error, and start both of the apply processes at `mult3.net`.

```
*/

BEGIN
  DBMS_APPLY_ADM.SET_PARAMETER(
    apply_name => 'apply_from_mult1',
    parameter  => 'disable_on_error',
    value      => 'n');
END;
/

BEGIN
  DBMS_APPLY_ADM.START_APPLY(
    apply_name => 'apply_from_mult1');
END;
/

BEGIN
  DBMS_APPLY_ADM.SET_PARAMETER(
    apply_name => 'apply_from_mult2',
    parameter  => 'disable_on_error',
    value      => 'n');
END;
/
```

```

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

/*

```

Step 26 Start the Apply Processes at mult1.net

Connect to mult1.net as the strmadmin user.

```

*/

CONNECT strmadmin/strmadminpw@mult1.net

/*

```

Set the `disable_on_error` parameter to `n` for both apply processes so that they will not be disabled if they encounter an error, and start both of the apply processes at mult1.net.

```

*/

BEGIN
    DBMS_APPLY_ADM.SET_PARAMETER(
        apply_name => 'apply_from_mult2',
        parameter  => 'disable_on_error',
        value      => 'n');
END;
/

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

BEGIN
    DBMS_APPLY_ADM.SET_PARAMETER(
        apply_name => 'apply_from_mult3',
        parameter  => 'disable_on_error',
        value      => 'n');
END;
/

```

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

/*
```

Step 27 Start the Capture Process at mult1.net

Start the capture process at mult1.net.

```
*/

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

/*
```

Step 28 Start the Capture Process at mult2.net

Connect to mult2.net as the strmadmin user.

```
*/

CONNECT strmadmin/strmadminpw@mult2.net

/*
```

Start the capture process at mult2.net.

```
*/

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

/*
```

Step 29 Start the Capture Process at mult3.net

Connect to mult3.net as the strmadmin user.

```
*/  
  
CONNECT strmadmin/strmadminpw@mult3.net  
  
/*
```

Start the capture process at mult3.net.

```
*/  
  
BEGIN  
  DBMS_CAPTURE_ADM.START_CAPTURE(  
    capture_name => 'capture_hr');  
END;  
/  
  
SET ECHO OFF  
  
/*
```

Step 30 Check the Spool Results

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

```
*/  
  
SET ECHO OFF  
SPOOL OFF  
  
/***** END OF SCRIPT *****/
```

Make DML and DDL Changes to Tables in the hr Schema

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

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

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

Step 1 Make a DML Change to `hr.employees` at `mult.net` and `mult2.net`

Make the following changes. Try to commit them at nearly the same time, but commit the change at `mult2.net` after you commit the change at `mult1.net`.

```
CONNECT hr/hr@mult1.net
```

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

```
CONNECT hr/hr@mult2.net
```

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

Step 2 Alter the `hr.jobs` Table at `mult3.net`

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

```
CONNECT hr/hr@mult3.net
```

```
ALTER TABLE hr.jobs RENAME COLUMN job_title TO job_name;
```


Step 3 Query the hr.employees Table at Each Database

After some time passes to allow for capture, propagation, and apply of the changes performed in Step 1, run the following query to confirm that the UPDATE changes have been applied at each database.

```
CONNECT hr/hr@mult1.net
```

```
SELECT salary FROM hr.employees WHERE employee_id=206;
```

```
CONNECT hr/hr@mult2.net
```

```
SELECT salary FROM hr.employees WHERE employee_id=206;
```

```
CONNECT hr/hr@mult3.net
```

```
SELECT salary FROM hr.employees WHERE employee_id=206;
```

All of the queries should show 10000 for the value of the salary.

Step 4 Describe the hr.jobs Table at Each Database

After some time passes to allow for capture, propagation, and apply of the change performed in Step 2, describe the hr . jobs table at each database to confirm that the ALTER TABLE change was propagated and applied correctly.

```
CONNECT hr/hr@mult1.net
```

```
DESC hr.jobs
```

```
CONNECT hr/hr@mult2.net
```

```
DESC hr.jobs
```

```
CONNECT hr/hr@mult3.net
```

```
DESC hr.jobs
```

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

Rule-Based Application Example

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

This chapter contains these topics:

- [Overview of the Rule-Based Application](#)
- [Using Rules on Non-Table Data Stored in Explicit Variables](#)
- [Using Rules on Data Stored in a Table](#)
- [Using Rules on Both Explicit Variables and Table Data](#)
- [Using Rules on Implicit Variables and Table Data](#)

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

See Also:

- [Chapter 5, "Rules"](#)
- [Chapter 15, "Managing Rules and Rule-Based Transformations"](#)
- ["Monitoring Rules and Rule-Based Transformations" on page 17-42](#)

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`.

Note: To complete these examples, the `COMPATIBLE` initialization parameter must be set to `9.2.0` or higher.

Using Rules on Non-Table Data Stored in Explicit Variables

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

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

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

Complete the following steps:

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

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the next "END OF SCRIPT" line on page 24-9 into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to all of the databases in the environment.

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

Step 1 Show Output and Spool Results

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

```
*/  
  
SET ECHO ON  
SPOOL rules_stored_variables.out  
  
/*
```

Step 2 Create the support User

```
*/  
  
CONNECT SYS/CHANGE_ON_INSTALL AS SYSDBA;  
  
GRANT CONNECT, RESOURCE TO support IDENTIFIED BY support;  
  
/*
```

Step 3 Grant the support User the Necessary System Privileges on Rules

```

*/

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

/*

```

Step 4 Create the evalctx Evaluation Context

```

*/

CONNECT support/support

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

```

```
/*
```

Step 5 Create the Rules that Correspond to Problem Priority

The following code creates one action context for each rule, and one name-value pair in each action context.

```
*/
```

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

/*
```


Step 6 Create the rs Rule Set

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

Step 7 Add the Rules to the Rule Set

```
*/  
  
BEGIN  
  DBMS_RULE_ADM.ADD_RULE(  
    rule_name      => 'r1',  
    rule_set_name => 'rs');  
  DBMS_RULE_ADM.ADD_RULE(  
    rule_name      => 'r2',  
    rule_set_name => 'rs');  
  DBMS_RULE_ADM.ADD_RULE(  
    rule_name      => 'r3',  
    rule_set_name => 'rs');  
END;  
/  
  
/*
```

Step 8 Query the Data Dictionary

At this point, you can view the evaluation context, rules, and rule set you created in the previous steps.

```
*/  
  
SELECT * FROM USER_EVALUATION_CONTEXTS;  
  
SELECT * FROM USER_RULES;  
  
SELECT * FROM USER_RULE_SETS;  
  
/*
```

Step 9 Create the problem_dispatch PL/SQL Procedure

```

*/

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

```

```
/*
```

Step 10 Dispatch Sample Problems

```
*/
```

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

```
/*
```

Step 11 Check the Spool Results

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

```
*/
```

```
SET ECHO OFF
SPOOL OFF
```

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

Using Rules on Data Stored in a Table

This example illustrates using rules to evaluate data stored in a table. This example is similar to the example described in ["Using Rules on Non-Table Data Stored in Explicit Variables"](#) on page 24-3. In both examples, the application routes customer problems based on priority. However, in this example, the problems are stored in a table instead of variables.

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

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

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

Complete the following steps:

1. [Show Output and Spool Results](#)
2. [Drop and Recreate the support User](#)
3. [Grant the support User the Necessary System Privileges on Rules](#)
4. [Create the problems Table](#)
5. [Create the evalctx Evaluation Context](#)
6. [Create the Rules that Correspond to Problem Priority](#)
7. [Create the rs Rule Set](#)
8. [Add the Rules to the Rule Set](#)
9. [Query the Data Dictionary](#)
10. [Create the problem_dispatch PL/SQL Procedure](#)
11. [Log Problems](#)
12. [List the Problems in the problems Table](#)
13. [Dispatch the Problems by Running the problem_dispatch Procedure](#)
14. [List the Problems in the problems Table](#)
15. [Check the Spool Results](#)

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the next "END OF SCRIPT" line on page 24-17 into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to all of the databases in the environment.

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

Step 1 Show Output and Spool Results

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

```
*/  
  
SET ECHO ON  
SPOOL rules_table.out  
  
/*
```

Step 2 Drop and Recreate the support User

```
*/  
  
CONNECT SYS/CHANGE_ON_INSTALL AS SYSDBA;  
  
DROP USER support CASCADE;  
  
GRANT CONNECT, RESOURCE TO support IDENTIFIED BY support;  
  
/*
```

Step 3 Grant the support User the Necessary System Privileges on Rules

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

Step 4 Create the problems Table

```
*/  
  
CONNECT support/support  
  
SET FEEDBACK 1  
SET NUMWIDTH 10  
SET LINESIZE 80  
SET TRIMSPOOL ON  
SET TAB OFF  
SET PAGESIZE 100  
SET SERVEROUTPUT ON;  
  
CREATE TABLE problems(  
    probid          NUMBER PRIMARY KEY,  
    custid          NUMBER,  
    priority        NUMBER,  
    description     VARCHAR2(4000),  
    center          VARCHAR2(100));  
  
/*
```

Step 5 Create the evalctx Evaluation Context

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

Step 6 Create the Rules that Correspond to Problem Priority

The following code creates one action context for each rule, and one name-value pair in each action context.

```

*/

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

/*

```

Step 7 Create the rs Rule Set

```

*/

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

```

```
/
```

```
/*
```

Step 8 Add the Rules to the Rule Set

```
*/
```

```
BEGIN
```

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

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

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

```
END;
```

```
/
```

```
/*
```

Step 9 Query the Data Dictionary

At this point, you can view the evaluation context, rules, and rule set you created in the previous steps.

```
*/
```

```
SELECT * FROM USER_EVALUATION_CONTEXTS;
```

```
SELECT * FROM USER_RULES;
```

```
SELECT * FROM USER_RULE_SETS;
```

```
/*
```

Step 10 Create the problem_dispatch PL/SQL Procedure

```
*/
```



```

CREATE OR REPLACE PROCEDURE problem_dispatch
IS
    cursor c IS SELECT probid, rowid FROM problems WHERE center IS NULL;
    tv      SYS.RE$TABLE_VALUE;
    tvl     SYS.RE$TABLE_VALUE_LIST;
    truehits SYS.RE$RULE_HIT_LIST;
    maybehits SYS.RE$RULE_HIT_LIST;
    ac      SYS.RE$NV_LIST;
    namearray SYS.RE$NAME_ARRAY;
    name    VARCHAR2(30);
    cval    VARCHAR2(100);
    rnum    INTEGER;
    i       INTEGER;
    status  PLS_INTEGER;
BEGIN
    FOR r IN c LOOP
        tv := SYS.RE$TABLE_VALUE('prob', rowidtochar(r.rowid));
        tvl := SYS.RE$TABLE_VALUE_LIST(tv);
        truehits := SYS.RE$RULE_HIT_LIST();
        maybehits := SYS.RE$RULE_HIT_LIST();
        DBMS_RULE.EVALUATE(
            rule_set_name      => 'support.rs',
            evaluation_context => 'evalctx',
            table_values       => tvl,
            true_rules         => truehits,
            maybe_rules        => maybehits);
        FOR rnum IN 1..truehits.COUNT LOOP
            DBMS_OUTPUT.PUT_LINE('Using rule ' || truehits(rnum).rule_name);
            ac := truehits(rnum).rule_action_context;
            namearray := ac.GET_ALL_NAMES;
            FOR i IN 1..namearray.COUNT LOOP
                name := namearray(i);
                status := ac.GET_VALUE(name).GETVARCHAR2(cval);
                IF (name = 'CENTER') THEN
                    UPDATE PROBLEMS SET center = cval WHERE rowid = r.rowid;
                    DBMS_OUTPUT.PUT_LINE('Assigning ' || r.probid || ' to ' || cval);
                    ELSIF (name = 'ALERT') THEN
                        DBMS_OUTPUT.PUT_LINE('Alert: ' || cval || ' Problem: ' || r.probid);
                    END IF;
                END LOOP;
            END LOOP;
        END LOOP;
    END LOOP;
END;
/

```

```
/*
```

Step 11 Log Problems

```
*/
```

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

```
INSERT INTO problems(probid, custid, priority, description)
VALUES(10102, 21, 2, 'noise on local calls');
```

```
INSERT INTO problems(probid, custid, priority, description)
VALUES(10103, 31, 3, 'noise on long distance calls');
```

```
COMMIT;
```

```
/*
```

Step 12 List the Problems in the problems Table

This `SELECT` statement should show the problems logged in Step 11. Notice that the center column is `NULL` for each new row inserted.

```
*/
```

```
SELECT * FROM problems;
```

```
/*
```

Step 13 Dispatch the Problems by Running the `problem_dispatch` Procedure

```
*/
```

```
EXECUTE problem_dispatch;
```

```
/*
```

Step 14 List the Problems in the problems Table

If the problems were dispatched successfully in Step 13, then this `SELECT` statement should show the center to which each problem was dispatched in the `center` column.

```
*/  
  
SELECT * FROM problems;  
  
/*
```

Step 15 Check the Spool Results

Check the `rules_table.out` spool file to ensure that all actions completed successfully after this script completes.

```
*/  
  
SET ECHO OFF  
SPOOL OFF  
  
/***** END OF SCRIPT *****/
```

Using Rules on Both Explicit Variables and Table Data

This example illustrates using rules to evaluate data stored in explicit variables and in a table. The application uses `problems` table in the `support` schema, into which customer problems are inserted. This example uses the following rules for handling customer problems:

- Assign all problems with priority greater than 2 to the San Jose Center
- Assign all problems with priority equal to 2 to the New York Center
- Assign all problems with priority equal to 1 to the Tampa Center from 8 AM to 8 PM
- Assign all problems with priority equal to 1 to the Bangalore Center from 8 PM to 8 AM
- Send an alert to the vice president of support for a problem with priority equal to 1

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

Some of the rules in this example refer to the current time, which is represented as an explicit variable named `current_time`. The current time is treated as additional data in the evaluation context. It is represented as a variable for the following reasons:

- It is not practical to store the current time in a table since it would have to be updated very often.
- The current time can be accessed by inserting calls to `SYSDATE` in every rule that requires it, but that would cause repeated invocations of the same SQL function `SYSDATE`, which may slow down rule evaluation. Different values of the current time in different rules may lead to incorrect behavior.

Complete the following steps:

1. [Show Output and Spool Results](#)
2. [Drop and Recreate the support User](#)
3. [Grant the support User the Necessary System Privileges on Rules](#)
4. [Create the problems Table](#)
5. [Create the evalctx Evaluation Context](#)

6. Create the Rules that Correspond to Problem Priority
7. Create the rs Rule Set
8. Add the Rules to the Rule Set
9. Query the Data Dictionary
10. Create the problem_dispatch PL/SQL Procedure
11. Log Problems
12. List the Problems in the problems Table
13. Dispatch the Problems by Running the problem_dispatch Procedure
14. List the Problems in the problems Table
15. Check the Spool Results

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the next "END OF SCRIPT" line on page 24-27 into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to all of the databases in the environment.

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

Step 1 Show Output and Spool Results

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

```
*/
```

```
SET ECHO ON  
SPOOL rules_var_tab.out
```

```
/*
```

Step 2 Drop and Recreate the support User

```
*/  
  
CONNECT SYS/CHANGE_ON_INSTALL AS SYSDBA;  
  
DROP USER support CASCADE;  
  
GRANT CONNECT, RESOURCE TO support IDENTIFIED BY support;  
  
/*
```

Step 3 Grant the support User the Necessary System Privileges on Rules

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

Step 4 Create the problems Table

```
*/  
  
CONNECT support/support  
  
SET FEEDBACK 1  
SET NUMWIDTH 10  
SET LINESIZE 80  
SET TRIMSPOOL ON  
SET TAB OFF  
SET PAGESIZE 100  
SET SERVEROUTPUT ON;
```

```

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

```

```

/*

```

Step 5 Create the evalctx Evaluation Context

```

*/

```

```

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

```

```

/*

```

Step 6 Create the Rules that Correspond to Problem Priority

The following code creates one action context for each rule, and one name-value pair in each action context.

```

*/

```

```
DECLARE
  ac SYS.RE$NV_LIST;
BEGIN
  ac := SYS.RE$NV_LIST(NULL);
  ac.ADD_PAIR('CENTER', SYS.AnyData.CONVERTVARCHAR2('San Jose'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name      => 'r1',
    condition      => 'prob.priority > 2',
    action_context => ac,
    rule_comment   => 'Low priority problems');
  ac := SYS.RE$NV_LIST(NULL);
  ac.ADD_PAIR('CENTER', SYS.AnyData.CONVERTVARCHAR2('New York'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name      => 'r2',
    condition      => 'prob.priority = 2',
    action_context => ac,
    rule_comment   => 'High priority problems');
  ac := SYS.RE$NV_LIST(NULL);
  ac.ADD_PAIR('ALERT', SYS.AnyData.CONVERTVARCHAR2('John Doe'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name      => 'r3',
    condition      => 'prob.priority = 1',
    action_context => ac,
    rule_comment   => 'Urgent problems');
  ac := SYS.RE$NV_LIST(NULL);
  ac.ADD_PAIR('CENTER', SYS.AnyData.CONVERTVARCHAR2('Tampa'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name => 'r4',
    condition => '(prob.priority = 1) and ' ||
                 '(TO_NUMBER(TO_CHAR(:current_time, ''HH24'')) >= 8) and ' ||
                 '(TO_NUMBER(TO_CHAR(:current_time, ''HH24'')) <= 20)',
    action_context => ac,
    rule_comment => 'Urgent daytime problems');
  ac := sys.RE$NV_LIST(NULL);
  ac.add_pair('CENTER', SYS.Anydata.CONVERTVARCHAR2('Bangalore'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name => 'r5',
    condition => '(prob.priority = 1) and ' ||
                 '((TO_NUMBER(TO_CHAR(:current_time, ''HH24'')) < 8) or ' ||
                 '(TO_NUMBER(TO_CHAR(:current_time, ''HH24'')) > 20))',
    action_context => ac,
    rule_comment => 'Urgent nighttime problems');
END;
/
```



```
/*
```

Step 7 Create the rs Rule Set

```
*/
```

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

```
/*
```

Step 8 Add the Rules to the Rule Set

```
*/
```

```
BEGIN
  DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'r1',
    rule_set_name => 'rs');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'r2',
    rule_set_name => 'rs');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'r3',
    rule_set_name => 'rs');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'r4',
    rule_set_name => 'rs');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'r5',
    rule_set_name => 'rs');
END;
/
```

```
/*
```

Step 9 Query the Data Dictionary

At this point, you can view the evaluation context, rules, and rule set you created in the previous steps.

```
*/  
  
SELECT * FROM USER_EVALUATION_CONTEXTS;  
  
SELECT * FROM USER_RULES;  
  
SELECT * FROM USER_RULE_SETS;  
  
/*
```

Step 10 Create the problem_dispatch PL/SQL Procedure

```
*/  
  
CREATE OR REPLACE PROCEDURE problem_dispatch  
IS  
    cursor c is SELECT probid, rowid FROM PROBLEMS WHERE center IS NULL;  
    tv        SYS.RE$TABLE_VALUE;  
    tvl       SYS.RE$TABLE_VALUE_LIST;  
    vv1       SYS.RE$VARIABLE_VALUE;  
    vv1       SYS.RE$VARIABLE_VALUE_LIST;  
    truehits  SYS.RE$RULE_HIT_LIST;  
    maybehits SYS.RE$RULE_HIT_LIST;  
    ac        SYS.RE$NV_LIST;  
    namearray SYS.RE$NAME_ARRAY;  
    name      VARCHAR2(30);  
    cval      VARCHAR2(100);  
    rnum      INTEGER;  
    i         INTEGER;  
    status    PLS_INTEGER;
```

```

BEGIN
  FOR r IN c LOOP
    tv := SYS.RE$TABLE_VALUE('prob', ROWIDTOCHAR(r.rowid));
    tvl := SYS.RE$TABLE_VALUE_LIST(tv);
    vv1 := SYS.RE$VARIABLE_VALUE('current_time',
                                  SYS.AnyData.CONVERTDATE(SYSDATE));
    vv1 := SYS.RE$VARIABLE_VALUE_LIST(vv1);
    truehits := SYS.RE$RULE_HIT_LIST();
    maybehits := SYS.RE$RULE_HIT_LIST();
    DBMS_RULE.EVALUATE(
      rule_set_name      => 'support.rs',
      evaluation_context => 'evalctx',
      table_values       => tvl,
      variable_values    => vv1,
      true_rules         => truehits,
      maybe_rules        => maybehits);
    FOR rnum IN 1..truehits.COUNT loop
      DBMS_OUTPUT.PUT_LINE('Using rule ' || truehits(rnum).rule_name);
      ac := truehits(rnum).rule_action_context;
      namearray := ac.GET_ALL_NAMES;
      FOR i in 1..namearray.COUNT LOOP
        name := namearray(i);
        status := ac.GET_VALUE(name).GETVARCHAR2(cval);
        IF (name = 'CENTER') THEN
          UPDATE problems SET center = cval
            WHERE rowid = r.rowid;
          DBMS_OUTPUT.PUT_LINE('Assigning ' || r.probid || ' to ' || cval);
        ELSIF (name = 'ALERT') THEN
          DBMS_OUTPUT.PUT_LINE('Alert: ' || cval || ' Problem: ' || r.probid);
        END IF;
      END LOOP;
    END LOOP;
  END LOOP;
END;
/

/*

```

Step 11 Log Problems

```
*/

INSERT INTO problems(probid, custid, priority, description)
VALUES(10201, 12, 1, 'no dial tone');

INSERT INTO problems(probid, custid, priority, description)
VALUES(10202, 22, 2, 'noise on local calls');

INSERT INTO PROBLEMS(probid, custid, priority, description)
VALUES(10203, 32, 3, 'noise on long distance calls');

COMMIT;

/*
```

Step 12 List the Problems in the problems Table

This `SELECT` statement should show the problems logged in Step 11. Notice that the center column is `NULL` for each new row inserted.

```
*/

SELECT * FROM problems;

/*
```

Step 13 Dispatch the Problems by Running the problem_dispatch Procedure

```
*/

EXECUTE problem_dispatch;

/*
```

Step 14 List the Problems in the problems Table

If the problems were dispatched successfully in Step 13, then this `SELECT` statement should show the center to which each problem was dispatched in the `center` column.

```
*/

SELECT * FROM problems;

/*
```

Step 15 Check the Spool Results

Check the `rules_var_tab.out` spool file to ensure that all actions completed successfully after this script completes.

```
*/

SET ECHO OFF
SPOOL OFF

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

Using Rules on Implicit Variables and Table Data

This example illustrates using rules to evaluate implicit variables and data stored in a table. The application uses the `problems` table in the `support` schema, into which customer problems are inserted. This example uses the following rules for handling customer problems:

- Assign all problems with priority greater than 2 to the San Jose Center
- Assign all problems with priority equal to 2 to the New York Center
- Assign all problems with priority equal to 1 to the Tampa Center from 8 AM to 8 PM
- Assign all problems with priority equal to 1 to the Bangalore Center from 8 PM to 8 AM
- Send an alert to the vice president of support for a problem with priority equal to 1

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

As in the example illustrated in "[Using Rules on Both Explicit Variables and Table Data](#)" on page 24-18, the current time is represented as a variable named `current_time`. However, this variable's value is not specified during evaluation by the caller. That is, `current_time` is an implicit variable in this example. A PL/SQL function named `timefunc` is specified for `current_time`, and this function is invoked once during evaluation to get its value.

Using implicit variables can be useful in other cases if one of the following conditions is true:

- The caller does not have access to the variable value
- The variable is referenced infrequently in rules. Because it is implicit, its value can be retrieved only when necessary, and does not need to be passed in for every evaluation.

Complete the following steps:

1. [Show Output and Spool Results](#)
2. [Drop and Recreate the support User](#)
3. [Grant the support User the Necessary System Privileges on Rules](#)
4. [Create the problems Table](#)
5. [Create the timefunc Function to Return the Value of current_time](#)
6. [Create the evalctx Evaluation Context](#)
7. [Create the Rules that Correspond to Problem Priority](#)
8. [Create the rs Rule Set](#)
9. [Add the Rules to the Rule Set](#)
10. [Query the Data Dictionary](#)
11. [Create the problem_dispatch PL/SQL Procedure](#)
12. [Log Problems](#)
13. [List the Problems in the problems Table](#)
14. [Dispatch the Problems by Running the problem_dispatch Procedure](#)
15. [List the Problems in the problems Table](#)
16. [Check the Spool Results](#)

Note: If you are viewing this document online, then you can copy the text from the "BEGINNING OF SCRIPT" line on this page to the next "END OF SCRIPT" line on page 24-36 into a text editor and then edit the text to create a script for your environment. Run the script with SQL*Plus on a computer that can connect to all of the databases in the environment.

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

Step 1 Show Output and Spool Results

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

```
*/  
  
SET ECHO ON  
SPOOL rules_implicit_var.out  
  
/*
```

Step 2 Drop and Recreate the support User

```
*/  
  
CONNECT SYS/CHANGE_ON_INSTALL AS SYSDBA;  
  
DROP USER support CASCADE;  
  
GRANT CONNECT, RESOURCE TO support IDENTIFIED BY support;  
  
/*
```

Step 3 Grant the support User the Necessary System Privileges on Rules

```
*/
```

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

/*
```

Step 4 Create the problems Table

```
*/

CONNECT support/support

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

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

/*
```

Step 5 Create the timefunc Function to Return the Value of current_time

```
*/
```



```

CREATE OR REPLACE FUNCTION timefunc(
    eco    VARCHAR2,
    ecn    VARCHAR2,
    var    VARCHAR2,
    evctx  SYS.RE$NV_LIST)
RETURN SYS.RE$VARIABLE_VALUE
IS
BEGIN
    IF (var = 'CURRENT_TIME') THEN
        RETURN(SYS.RE$VARIABLE_VALUE('CURRENT_TIME',
                                     SYS.AnyData.CONVERTDATE(sysdate)));
    ELSE
        RETURN(NULL);
    END IF;
END;
/

/*

```

Step 6 Create the evalctx Evaluation Context

```

*/

DECLARE
    ta SYS.RE$TABLE_ALIAS_LIST;
    vt SYS.RE$VARIABLE_TYPE_LIST;
BEGIN
    ta := SYS.RE$TABLE_ALIAS_LIST(SYS.RE$TABLE_ALIAS('prob', 'problems'));
    vt := SYS.RE$VARIABLE_TYPE_LIST(
        SYS.RE$VARIABLE_TYPE('current_time', 'DATE', 'timefunc', NULL));
    DBMS_RULE_ADM.CREATE_EVALUATION_CONTEXT(
        evaluation_context_name => 'evalctx',
        table_aliases           => ta,
        variable_types          => vt,
        evaluation_context_comment => 'support problem definition');
END;
/

/*

```

Step 7 Create the Rules that Correspond to Problem Priority

The following code creates one action context for each rule, and one name-value pair in each action context.

```

*/

```

```
DECLARE
  ac SYS.RE$NV_LIST;
BEGIN
  ac := SYS.RE$NV_LIST(NULL);
  ac.ADD_PAIR('CENTER', SYS.AnyData.CONVERTVARCHAR2('San Jose'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name      => 'r1',
    condition      => 'prob.priority > 2',
    action_context => ac,
    rule_comment   => 'Low priority problems');
  ac := SYS.RE$NV_LIST(NULL);
  ac.ADD_PAIR('CENTER', SYS.AnyData.CONVERTVARCHAR2('New York'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name      => 'r2',
    condition      => 'prob.priority = 2',
    action_context => ac,
    rule_comment   => 'High priority problems');
  ac := SYS.RE$NV_LIST(NULL);
  ac.ADD_PAIR('ALERT', SYS.AnyData.CONVERTVARCHAR2('John Doe'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name      => 'r3',
    condition      => 'prob.priority = 1',
    action_context => ac,
    rule_comment   => 'Urgent problems');
  ac := SYS.RE$NV_LIST(NULL);
  ac.ADD_PAIR('CENTER', SYS.AnyData.CONVERTVARCHAR2('Tampa'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name => 'r4',
    condition => '(prob.priority = 1) and ' ||
                 '(TO_NUMBER(TO_CHAR(:current_time, 'HH24')) >= 8) and ' ||
                 '(TO_NUMBER(TO_CHAR(:current_time, 'HH24')) <= 20)',
    action_context => ac,
    rule_comment   => 'Urgent daytime problems');
  ac := SYS.RE$NV_LIST(NULL);
  ac.add_pair('CENTER', sys.anydata.convertvarchar2('Bangalore'));
  DBMS_RULE_ADM.CREATE_RULE(
    rule_name => 'r5',
    condition => '(prob.priority = 1) and ' ||
                 '((TO_NUMBER(TO_CHAR(:current_time, 'HH24')) < 8) or ' ||
                 '(TO_NUMBER(TO_CHAR(:current_time, 'HH24')) > 20))',
    action_context => ac,
    rule_comment   => 'Urgent nighttime problems');
END;
/
```

```
/*
```

Step 8 Create the rs Rule Set

```
*/
```

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

```
/*
```

Step 9 Add the Rules to the Rule Set

```
*/
```

```
BEGIN
  DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'r1',
    rule_set_name => 'rs');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'r2',
    rule_set_name => 'rs');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'r3',
    rule_set_name => 'rs');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'r4',
    rule_set_name => 'rs');
  DBMS_RULE_ADM.ADD_RULE(
    rule_name      => 'r5',
    rule_set_name => 'rs');
END;
/
```

```
/*
```

Step 10 Query the Data Dictionary

At this point, you can view the evaluation context, rules, and rule set you created in the previous steps.

```
*/  
  
SELECT * FROM USER_EVALUATION_CONTEXTS;  
  
SELECT * FROM USER_RULES;  
  
SELECT * FROM USER_RULE_SETS;  
  
/*
```

Step 11 Create the problem_dispatch PL/SQL Procedure

```
*/  
  
CREATE OR REPLACE PROCEDURE problem_dispatch  
IS  
    cursor c IS SELECT probid, rowid FROM problems WHERE center IS NULL;  
    tv      SYS.RE$TABLE_VALUE;  
    tvl     SYS.RE$TABLE_VALUE_LIST;  
    truehits SYS.RE$RULE_HIT_LIST;  
    maybehits SYS.RE$RULE_HIT_LIST;  
    ac      SYS.RE$NV_LIST;  
    namearray SYS.RE$NAME_ARRAY;  
    name    VARCHAR2(30);  
    cval    VARCHAR2(100);  
    rnum    INTEGER;  
    i       INTEGER;  
    status  PLS_INTEGER;  
BEGIN  
    FOR r IN c LOOP  
        tv := SYS.RE$TABLE_VALUE('prob', rowidtochar(r.rowid));  
        tvl := SYS.RE$TABLE_VALUE_LIST(tv);  
        truehits := SYS.RE$RULE_HIT_LIST();  
        maybehits := SYS.RE$RULE_HIT_LIST();  
        DBMS_RULE.EVALUATE(  
            rule_set_name => 'support.rs',  
            evaluation_context => 'evalctx',  
            table_values => tvl,  
            true_rules => truehits,  
            maybe_rules => maybehits);  
    END LOOP;  
END;
```

```

FOR rnum IN 1..truehits.COUNT LOOP
  DBMS_OUTPUT.PUT_LINE('Using rule ' || truehits(rnum).rule_name);
  ac := truehits(rnum).rule_action_context;
  namearray := ac.GET_ALL_NAMES;
  FOR i IN 1..namearray.COUNT LOOP
    name := namearray(i);
    status := ac.GET_VALUE(name).GETVARCHAR2(cval);
    IF (name = 'CENTER') THEN
      UPDATE problems SET center = cval
        WHERE rowid = r.rowid;
      DBMS_OUTPUT.PUT_LINE('Assigning ' || r.probid || ' to ' || cval);
    ELSIF (name = 'ALERT') THEN
      DBMS_OUTPUT.PUT_LINE('Alert: ' || cval || ' Problem: ' || r.probid);
    END IF;
  END LOOP;
END LOOP;
END LOOP;
END;
/

/*

```

Step 12 Log Problems

```

*/

INSERT INTO problems(probid, custid, priority, description)
VALUES(10301, 13, 1, 'no dial tone');

INSERT INTO problems(probid, custid, priority, description)
VALUES(10302, 23, 2, 'noise on local calls');

INSERT INTO problems(probid, custid, priority, description)
VALUES(10303, 33, 3, 'noise on long distance calls');

COMMIT;

/*

```

Step 13 List the Problems in the problems Table

This `SELECT` statement should show the problems logged in Step 12. Notice that the center column is `NULL` for each new row inserted.

```
*/  
  
SELECT * FROM problems;  
  
/*
```

Step 14 Dispatch the Problems by Running the problem_dispatch Procedure

```
*/  
  
EXECUTE problem_dispatch;  
  
/*
```

Step 15 List the Problems in the problems Table

If the problems were dispatched successfully in Step 13, then this `SELECT` statement should show the center to which each problem was dispatched in the `center` column.

```
*/  
  
SELECT * FROM problems;  
  
/*
```

Step 16 Check the Spool Results

Check the `rules_implicit_var.out` spool file to ensure that all actions completed successfully after this script completes.

```
*/  
  
SET ECHO OFF  
SPOOL OFF  
  
/***** END OF SCRIPT *****/
```

Part IV

Appendixes

This part includes the following appendix:

- [Appendix A, "XML Schema for LCRs"](#)

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/streams1cr.xsd
```

This schema definition can be loaded into the database by connecting as SYS in SQL*Plus and executing the following file:

```
rdbms/admin/catx1cr.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="VARCHAR2"
        nillable="true"/>

      <!-- Represent char as varchar2. xdb:CHAR blank pads upto 2000 bytes! -->
      <element name="char" type = "string" xdb:SQLType="VARCHAR2"
        nillable="true"/>
    </choice>
  </complexType>
</schema>
```

```

<element name="nchar" type = "string" xdb:SQLType="NVARCHAR2"
    nillable="true"/>
<element name="nvarchar2" type = "string" xdb:SQLType="NVARCHAR2"
    nillable="true"/>
<element name="number" type = "double" xdb:SQLType="NUMBER"
    nillable="true"/>
<element name="raw" type = "hexBinary" xdb:SQLType="RAW"
    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"/>
</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"/>
    </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"/>
        </sequence>
    </complexType>
</element>

```

```

<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>
</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" 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"/>
    </sequence>
  </complexType>
</element>

```

```
<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"/>
</sequence>
</complexType>
</element>
</schema>';
```


A

- ABORT_GLOBAL_INSTANTIATION
 - procedure, 12-12
- ABORT_SCHEMA_INSTANTIATION
 - procedure, 12-12
- ABORT_TABLE_INSTANTIATION
 - procedure, 12-12
- action contexts, 5-9
 - adding name-value pairs, 15-16, 15-20
 - creating
 - example, 22-35
 - querying, 15-15
 - removing name-value pairs, 15-20
 - system-created rules, 6-18
- ADD_SUPPLEMENTAL_LOG_DATA clause, 12-10
- ADD_SUPPLEMENTAL_LOG_GROUP clause, 12-9, 20-12, 21-12, 22-23, 23-26
- ADD_COLUMN member procedure, 20-16
- ADD_GLOBAL_RULES procedure, 6-14
- ADD_PAIR member procedure, 15-16, 15-20, 22-35, 24-6, 24-13, 24-31
- ADD_RULE procedure, 5-8, 15-5
- ADD_SCHEMA_PROPAGATION_RULES procedure, 6-12
- ADD_SUBSCRIBER procedure, 13-3, 19-25, 20-15
- ADD_SUBSET_RULES procedure, 4-12, 6-5, 6-6
 - row migration, 4-12
- ADD_TABLE_RULES procedure, 6-6
- alert log
 - Oracle Streams entries, 18-24
- ALL_STREAMS_GLOBAL_RULES view, 17-44
- ALL_STREAMS_SCHEMA_RULES view, 17-44
- ALL_STREAMS_TABLE_RULES view, 17-44
- ALTER_DATABASE statement
 - ADD_SUPPLEMENTAL_LOG_DATA clause, 12-10
 - DROP_SUPPLEMENTAL_LOG_DATA clause, 12-10
- ALTER_TABLE statement
 - ADD_SUPPLEMENTAL_LOG_GROUP clause, 12-9, 20-12, 21-12, 22-23, 23-26
 - DROP_SUPPLEMENTAL_LOG_GROUP clause, 12-10
- ALTER_APPLY procedure
 - removing the DDL handler, 14-21
 - removing the message handler, 14-13
 - removing the rule set, 14-10
 - removing the tag value, 16-28
 - setting an apply user, 14-12
 - setting the DDL handler, 14-20
 - setting the message handler, 14-13
 - setting the tag value, 8-2, 8-6, 16-28
 - specifying the rule set, 14-8
- ALTER_CAPTURE procedure
 - removing the rule set, 12-7
 - setting the start SCN, 12-11
 - specifying a rule set, 12-5
- ALTER_PROPAGATION procedure
 - removing the rule set, 13-16
 - specifying the rule set, 13-14
- ALTER_PROPAGATION_SCHEDULE procedure, 13-12
- ALTER_RULE procedure, 15-5
- AnyData datatype
 - message propagation, 3-16

- queues, 3-11, 13-18
 - creating, 13-2
 - dequeuing, 13-22
 - dropping, 13-7
 - enqueueing, 13-19
 - monitoring, 17-12
 - propagating to typed queues, 3-16
 - user-defined types, 3-17
 - wrapper for messages, 3-11, 13-19
- applied SCN, 2-16, 4-28, 17-7
- apply forwarding, 3-8
- apply process, 4-1
 - applied SCN, 4-28
 - apply forwarding, 3-8
 - apply handlers, 4-16
 - apply servers, 4-31
 - apply user, 4-2
 - setting, 14-12
- architecture, 4-30
- automatic restart, 4-36
- conflict handlers, 4-16
 - heterogeneous environments, 9-5
- conflict resolution, 4-16, 7-1
- constraints, 4-10
- coordinator process, 4-31
- creating, 14-2
- creation, 4-32
- datatypes applied, 4-9
 - heterogeneous environments, 9-6
- DDL changes, 4-21
 - CREATE TABLE AS SELECT, 4-24
 - current schema, 4-23
 - data structures, 4-22
 - ignored, 4-21
 - system-generated names, 4-23
- DDL handlers, 4-4
 - creating, 14-19
 - monitoring, 17-24
 - removing, 14-21
 - setting, 14-20
- dependent transactions, 18-12
- DML changes, 4-10
 - heterogeneous environments, 9-7
- DML handlers, 4-4
 - creating, 14-14, 20-16
 - heterogeneous environments, 9-5
 - monitoring, 17-23
 - re-enqueue captured events, 20-1
 - setting, 14-16
- dropping, 14-7
- error handlers
 - creating, 14-21
 - heterogeneous environments, 9-5
 - monitoring, 17-23
 - setting, 14-26
- events, 4-3
 - captured, 4-3
 - user-enqueued, 4-3
- exception queue, 4-37
 - monitoring, 17-36, 17-37
- heterogeneous environments, 9-3, 9-12
 - database links, 9-4
 - example, 22-38
- high-watermark, 4-28
- ignore SCN, 4-27
- instantiation SCN, 4-27
- key columns, 4-10
- LOBs, 16-13
- logical change records (LCRs), 4-4
- low-watermark, 4-28
- managing, 14-1
- message handlers, 4-4
 - creating, 19-17
 - heterogeneous environments, 9-5
 - monitoring, 17-24
 - removing, 14-13
 - setting, 14-13
- monitoring, 17-20
 - apply handlers, 17-23
 - latency, 17-29, 17-31
- non-LCR events, 4-7
- oldest SCN, 4-28
- options, 4-4
- Oracle Real Application Clusters, 4-29
- parallelism, 17-35
- parameters, 4-34
 - commit_serialization, 4-35, 18-12
 - disable_on_error, 4-36

- disable_on_limit, 4-36
- heterogeneous environments, 9-5
- parallelism, 4-34, 18-12
- setting, 14-11
- time_limit, 4-36
- transaction_limit, 4-36
- persistent state, 4-36
- reader server, 4-31
- RESTRICTED SESSION, 4-29
- row migration, 4-12
- row subsetting, 4-12, 6-5
 - supplemental logging, 4-14
- rule set
 - removing, 14-10
 - specifying, 14-8
- rules, 4-2, 6-2
 - adding, 14-8
 - removing, 14-9
- starting, 14-7
- stopping, 14-7
- substitute key columns, 4-11
 - heterogeneous environments, 9-4, 9-5
 - removing, 14-29
 - setting, 14-28
- tables, 4-10
 - apply handlers, 4-16
 - column discrepancies, 4-14
- tags, 8-6
 - monitoring, 17-51
 - removing, 16-28
 - setting, 16-28
- trace files, 18-26
- transformations
 - rule-based, 6-32
- triggers
 - firing property, 4-25
 - ON SCHEMA clause, 4-26
- troubleshooting, 18-9
 - checking apply handlers, 18-12
 - checking event type, 18-11
 - checking status, 18-10
 - exception queue, 18-13
- AQ_TM_PROCESSES initialization
 - parameter, 11-5, 21-3, 22-5

- ARCHIVE_LAG_TARGET initialization
 - parameter, 11-5
- ARCHIVELOG mode
 - capture process, 2-25, 11-13, 20-4, 21-3, 22-6

B

- backups
 - online
 - Streams, 8-5

C

- capture process, 2-1
 - applied SCN, 2-16, 17-7
 - architecture, 2-18
 - ARCHIVELOG mode, 2-25, 11-13, 20-4, 21-3, 22-6
 - automatic restart, 2-26
 - builder server, 2-19
 - captured events, 3-3
 - captured SCN, 2-16
 - changes captured, 2-7
 - DDL changes, 2-8
 - DML changes, 2-7
 - NOLOGGING keyword, 2-9
 - UNRECOVERABLE clause for SQL*Loader, 2-10
 - UNRECOVERABLE keyword, 2-9
 - configuring, 11-13
 - creating, 12-2
 - creation, 2-21
 - data dictionary duplication, 2-22
 - datatypes captured, 2-6
 - DBID, 2-22, 12-13
 - DBNAME, 2-22
 - dropping, 12-14
 - heterogeneous environments, 9-3
 - log sequence number, 12-13
 - LogMiner, 2-20
 - alternate tablespace for, 2-20, 11-13
 - multiple sessions, 2-20
 - LOGMNR_MAX_PERSISTENT_SESSIONS
 - initialization parameter, 2-20
 - managing, 12-1

- monitoring, 17-3
 - applied SCN, 17-7
 - latency, 17-7, 17-8
- Oracle Real Application Clusters, 2-17
- parameters, 2-25
 - disable_on_limit, 2-26
 - message_limit, 2-26
 - parallelism, 2-26
 - setting, 12-8
 - time_limit, 2-26
- persistent state, 2-30
- preparer servers, 2-19
- reader server, 2-19
- redo logs, 2-2
- re-enqueue events, 20-1
- RESTRICTED SESSION, 2-16
- rule evaluation, 2-27
- rule set
 - removing, 12-7
 - specifying, 12-5
- rules, 2-5, 6-2
 - adding, 12-5
 - removing, 12-6
- start SCN, 2-15, 2-16
 - setting, 12-11
- starting, 12-5
- stopping, 12-14
- supplemental logging, 2-11
 - specifying, 12-9
- SYS schema, 2-6, 2-7
- SYSTEM schema, 2-6, 2-7
- trace files, 18-25
- transformations
 - rule-based, 6-28
- troubleshooting, 18-2
 - checking progress, 18-3
 - checking status, 18-2
 - log parallelism, 18-3
 - persistent sessions, 18-4
- captured SCN, 2-16
- change cycling
 - avoidance
 - tags, 8-8
- column lists, 7-12
- COMPATIBLE initialization parameter, 11-5, 19-4, 20-4, 21-3, 22-5
- conditions
 - rules, 5-2
- conflict resolution, 7-1
 - column lists, 7-12
 - conflict handlers, 7-4, 7-5, 7-6, 7-8
 - custom, 7-15
 - interaction with apply handlers, 4-16
 - modifying, 14-31
 - prebuilt, 7-8
 - removing, 14-32
 - setting, 14-30
 - data convergence, 7-15
 - DISCARD handler, 7-10
 - MAXIMUM handler, 7-10
 - example, 23-32
 - latest time, 7-11
 - MINIMUM handler, 7-12
 - OVERWRITE handler, 7-10
 - resolution columns, 7-14
 - time-based, 7-11
 - example, 23-32
 - preparing for, 23-12
- conflicts
 - avoidance, 7-6
 - delete, 7-7
 - primary database ownership, 7-6
 - uniqueness, 7-6
 - update, 7-7
 - delete, 7-3
 - detection, 7-5
 - identifying rows, 7-5
 - DML conflicts, 7-2
 - foreign key, 7-3
 - transaction ordering, 7-4
 - types of, 7-2
 - uniqueness, 7-3
 - update, 7-3
- CONVERT_ANYDATA_TO_LCR_DDL
 - function, 13-28
- CONVERT_ANYDATA_TO_LCR_ROW
 - function, 13-28

CREATE TABLE statement
AS SELECT
 apply process, 4-24
CREATE_APPLY procedure, 4-32, 14-2
 tags, 8-2, 8-6
CREATE_CAPTURE procedure, 2-21, 12-2, 12-4
CREATE_EVALUATION_CONTEXT
 procedure, 24-5, 24-12, 24-21, 24-31
CREATE_PROPAGATION procedure, 13-8
CREATE_RULE procedure, 15-3
CREATE_RULE_SET procedure, 15-2

D

database links

Oracle Streams, 11-14

datatypes

applied, 4-9

captured, 2-6

heterogeneous environments, 9-6

DBA_APPLY view, 17-21, 17-24, 17-28, 17-33,
17-51, 18-10, 18-11

DBA_APPLY_CONFLICT_COLUMNS view, 17-26

DBA_APPLY_DML_HANDLERS view, 17-23

DBA_APPLY_ERROR view, 17-36, 17-37, 17-40,
17-41

DBA_APPLY_INSTANTIATED_OBJECTS
view, 17-27

DBA_APPLY_KEY_COLUMNS view, 17-25

DBA_APPLY_PARAMETERS view, 17-22

DBA_APPLY_PROGRESS view, 17-31

DBA_CAPTURE view, 17-3, 17-7, 18-2

DBA_CAPTURE_PARAMETERS view, 17-6

DBA_CAPTURE_PREPARED_DATABASE
view, 17-9

DBA_CAPTURE_PREPARED_SCHEMAS
view, 17-9

DBA_CAPTURE_PREPARED_TABLES view, 17-9

DBA_EVALUATION_CONTEXT_TABLES
view, 17-45

DBA_EVALUATION_CONTEXT_VARS
view, 17-46

DBA_LOG_GROUPS view, 17-11

DBA_PROPAGATION view, 17-15, 17-16, 17-17,
17-19, 18-5, 18-6

DBA_QUEUE_SCHEDULES view, 17-17, 17-19,
18-6

DBA_QUEUE_TABLES view, 17-12

DBA_QUEUES view, 17-12

DBA_RULE_SET_RULES view, 17-47, 17-48, 17-49

DBA_RULE_SETS view, 17-45

DBA_RULES view, 17-47, 17-48, 17-49

DBA_STREAMS_GLOBAL_RULES view, 17-44,
18-19

DBA_STREAMS_SCHEMA_RULES view, 17-44,
18-19, 18-21

DBA_STREAMS_TABLE_RULES view, 17-44,
18-18, 18-19

DBID (database identifier)

capture process, 2-22

DBMS_APPLY_ADM package, 14-1

DBMS_CAPTURE_ADM package, 12-1

DBMS_PROPAGATION_ADM package, 13-1

DBMS_RULE package, 5-12, 24-1

DBMS_RULE_ADM package, 15-2, 24-1

DBMS_STREAMS package, 16-26

DBMS_STREAMS_ADM package, 6-3, 12-1, 13-1,
14-1

apply process creation, 4-32

capture process creation, 2-21

creating a capture process, 12-2

creating a propagation, 13-8

creating an apply process, 14-2

tags, 8-3

DBMS_TRANSFORM package, 13-26, 13-29

DBNAME

capture process, 2-22

DDL handlers, 4-4

creating, 14-19

monitoring, 17-24

removing, 14-21

setting, 14-20

DELETE_ALL_ERRORS procedure, 14-35

DELETE_ERROR procedure, 4-37, 14-34

DEQUEUE procedure, 13-22

example, 19-25, 20-18

destination queue, 3-2

direct path load

capture processes, 2-10

- directed networks, 3-7
 - apply forwarding, 3-8
 - queue forwarding, 3-8
- DISABLE_DB_ACCESS procedure, 13-5
- DISABLE_PROPAGATION_SCHEDULE procedure, 13-17
- DISCARD conflict resolution handler, 7-10
- DML handlers, 4-4, 4-16
 - creating, 14-14, 20-16
 - monitoring, 17-23
 - removing, 14-18
 - setting, 14-16
- DROP SUPPLEMENTAL LOG DATA clause, 12-10
- DROP SUPPLEMENTAL LOG GROUP clause, 12-10
- DROP_APPLY procedure, 14-7
- DROP_CAPTURE procedure, 12-14
- DROP_PROPAGATION procedure, 13-18
- DROP_RULE procedure, 15-7
- DROP_RULE_SET procedure, 15-8

E

- ENABLE_DB_ACCESS procedure, 13-3
- ENABLE_PROPAGATION_SCHEDULE procedure, 13-11
- ENQUEUE procedure, 13-20, 16-4, 19-12, 20-16
- error handlers, 4-16
 - creating, 14-21
 - monitoring, 17-23
 - removing, 14-27
 - setting, 14-26
- EVALUATE procedure, 5-12, 24-8, 24-14, 24-24, 24-34
- evaluation contexts, 5-5
 - association with rule sets, 5-8
 - association with rules, 5-8
 - creating, 19-19, 24-5, 24-12, 24-21, 24-31
 - evaluation function, 5-8
 - object privileges
 - granting, 15-9
 - revoking, 15-10

- system privileges
 - granting, 15-9
 - revoking, 15-10
- user-created, 6-19, 6-24
- variables, 5-6
- event contexts
 - system-created rules, 6-18
- events
 - apply process, 4-3
 - captured, 3-3
 - propagating, 13-28
 - dequeue, 3-3
 - programmatically environments, 3-12
 - enqueue, 3-3
 - programmatically environments, 3-12
 - propagation, 3-4
 - user-enqueued, 3-3
 - propagating, 13-24
- exception queues, 4-37
 - apply process, 18-13
 - deleting errors, 14-34
 - executing errors, 14-33
 - heterogeneous environments, 9-10
 - monitoring, 17-36, 17-37
- EXECUTE member procedure, 14-15, 14-20, 14-24, 20-16
- EXECUTE_ALL_ERRORS procedure, 14-34
- EXECUTE_ERROR procedure, 4-37, 14-33
- Export
 - OBJECT_CONSISTENT parameter, 11-9, 11-10, 21-14, 22-27, 23-48
 - Oracle Streams, 11-8, 14-36, 16-35

G

- GET_ALL_NAMES member function, 24-8, 24-14, 24-24, 24-34
- GET_BASE_TABLE_NAME member function, 14-20
- GET_BASE_TABLE_OWNER member function, 14-20
- GET_COMMAND_TYPE member function, 14-20, 14-24, 17-39, 20-16
- GET_CURRENT_SCHEMA member function, 14-20

GET_DDL_TEXT member function, 17-39
 GET_ERROR_MESSAGE function, 17-40, 17-41
 GET_INFORMATION function, 14-24
 GET_LOB_INFORMATION member function, 16-8
 use_old parameter, 16-8
 GET_LOGON_USER member function, 14-20
 GET_OBJECT_NAME member function, 14-15, 14-20, 14-24, 15-12, 17-39, 22-34
 GET_OBJECT_OWNER member function, 14-15, 14-20, 15-12, 17-39
 GET_SCN member function, 14-15, 14-20
 GET_SOURCE_DATABASE_NAME member function, 14-20, 17-39
 GET_TAG function, 16-27, 17-50
 GET_TAG member function, 14-15, 14-20
 GET_TRANSACTION_ID member function, 14-15, 14-20
 GET_VALUE member function
 LCRs, 15-12, 16-9
 use_old parameter, 16-8
 rules, 24-8, 24-14, 24-24, 24-34
 GET_VALUES member function, 14-15, 14-24, 16-10, 17-39, 20-16
 use_old parameter, 16-8
 GLOBAL_NAMES initialization parameter, 11-5, 19-4, 20-4, 21-3, 22-5
 GLOBAL_NAMES view, 18-5
 GRANT_OBJECT_PRIVILEGE procedure, 5-15
 GRANT_SYSTEM_PRIVILEGE procedure, 5-15

H

heterogeneous information sharing, 9-1
 non-Oracle to non-Oracle, 9-13
 non-Oracle to Oracle, 9-10
 apply process, 9-12
 capturing changes, 9-11
 instantiation, 9-12
 user application, 9-11
 Oracle to non-Oracle, 9-2
 apply process, 9-3
 capture process, 9-3
 conflict handlers, 9-5
 database links, 9-4

 datatypes applied, 9-6
 DML changes, 9-7
 DML handlers, 9-5
 error handlers, 9-5
 errors, 9-10
 instantiation, 9-7
 message handlers, 9-5
 parallelism, 9-5
 staging, 9-3
 substitute key columns, 9-4, 9-5
 transformations, 9-9
 high availability
 Streams, 10-1
 advantages, 10-3
 apply, 10-11
 best practices, 10-7
 capture, 10-10
 database links, 10-9
 propagation, 10-11
 high-watermark, 4-28

I

ignore SCN, 4-27
 Import
 Oracle Streams, 11-8, 14-36, 16-35
 STREAMS_CONFIGURATION
 parameter, 11-10
 STREAMS_INSTANTIATION
 parameter, 11-10, 21-15, 22-28, 23-49
 initialization parameters
 AQ_TM_PROCESSES, 11-5
 ARCHIVE_LAG_TARGET, 11-5
 COMPATIBLE, 11-5
 GLOBAL_NAMES, 11-5
 JOB_QUEUE_PROCESSES, 11-6
 LOG_PARALLELISM, 11-6
 LOGMNR_MAX_PERSISTENT_SESSIONS,
 11-6
 OPEN_LINKS, 11-6
 Oracle Streams, 11-4
 PARALLEL_MAX_SERVERS, 11-7
 PROCESSES, 11-7
 SESSIONS, 11-7

- SGA_MAX_SIZE, 11-7
- SHARED_POOL_SIZE, 11-8
- instantiation
 - aborting preparation, 12-12
 - example, 21-14, 22-27, 22-48, 23-48
 - heterogeneous environments
 - non-Oracle to Oracle, 9-12
 - Oracle to non-Oracle, 9-7
 - Oracle Streams, 11-8, 14-36
 - preparing for, 2-13, 11-15, 12-11
 - setting an SCN, 11-15, 14-35
 - DDL LCRs, 14-38
 - export/import, 14-36
 - supplemental logging specifications, 2-15
- instantiation SCN, 4-27
- IS_NULL_TAG member function, 6-8, 17-39
- IS_TRIGGER_FIRE_ONCE function, 4-25

J

- JMS
 - Oracle Streams
 - example, 19-35
- job queue processes
 - propagation jobs, 3-20
- JOB_QUEUE_PROCESSES initialization
 - parameter, 11-6, 21-3, 22-5
 - propagation, 18-7

L

- LCR\$_ROW_UNIT type
 - GET_LOB_INFORMATION member
 - function, 16-8
- LCRs. *See* logical change records
- LOBs
 - Oracle Streams, 16-11
 - apply process, 16-13
 - constructing, 16-14
 - requirements, 16-12
- LOG_PARALLELISM initialization
 - parameter, 11-6, 20-4, 21-3, 22-5
 - capture process, 18-3

- logical change records (LCRs), 2-2
 - apply process, 4-4
 - constructing, 16-2
 - example, 19-13
 - DDL LCRs, 2-4
 - current_schema, 4-23
 - rules, 6-9
 - determining if tag is NULL, 6-8
 - enqueueing, 16-2
 - getting constraint, 14-24
 - getting information about, 14-15, 14-20, 15-12, 17-39
 - getting sender, 14-24
 - row LCRs, 2-3
 - getting column value, 16-9
 - getting list of column values, 14-24, 16-10
 - rules, 6-7
 - setting list of column values, 14-24
 - XML schema, A-1
- LogMiner
 - alternate tablespace for, 2-20, 11-13
 - capture process, 2-20
 - multiple sessions, 2-20
- LOGMNR_MAX_PERSISTENT_SESSIONS
 - initialization parameter, 2-20, 11-6
 - capture process, 18-4
- low-watermark, 4-28

M

- MAXIMUM conflict resolution handler, 7-10
 - latest time, 7-11
- message handlers, 4-4
 - creating, 19-17
 - monitoring, 17-24
- messaging
 - Oracle Streams, 19-1
 - propagation, 3-16
- MINIMUM conflict resolution handler, 7-12
- monitoring
 - AnyData datatype queues, 17-12
 - event consumers, 17-13
 - viewing event contents, 17-13

- apply process, 17-20
 - apply handlers, 17-23
 - DDL handlers, 17-24
 - error handlers, 17-23
 - exception queue, 17-36, 17-37
 - message handlers, 17-24
- capture process, 17-3
 - applied SCN, 17-7
 - latency, 17-7, 17-8
- DML handlers, 17-23
- Oracle Streams, 17-1
- propagation jobs, 17-15
- propagations, 17-15
- rule-based transformations
 - procedures, 17-49
- rules, 17-42
- tags, 17-50
 - apply process value, 17-51
 - current session value, 17-50

N

NOLOGGING mode

- capture process, 2-9

O

OBJECT_CONSISTENT parameter

- for Export utility, 11-9, 11-10, 21-14, 22-27, 23-48

oldest SCN, 4-28

ON SCHEMA clause

- of CREATE TRIGGER
 - apply process, 4-26

OPEN_LINKS initialization parameter, 11-6

ORA-01403 error, 18-14

ORA-24093 error, 18-8

ORA-25224 error, 18-9

ORA-26687 error, 18-15

ORA-26688 error, 18-15

ORA-26689 error, 18-16

Oracle Enterprise Manager

- Streams tool, 1-23

Oracle Real Application Clusters

- interoperation with Oracle Streams, 2-17, 3-18, 4-29

Oracle Streams

- adding databases, 11-22, 11-35, 22-76
- adding objects, 11-19, 11-30, 22-64
- administrator
 - creating, 11-2
- alert log, 18-24
- AnyData queues, 13-18
- apply process, 4-1
- capture process, 2-1
- configuring, 11-15
- conflict resolution, 7-1
- data dictionary, 2-22, 3-25, 4-33
- data dictionary views, 17-1
- database links, 11-14
- directed networks, 3-7
- example environments
 - messaging, 19-1
 - replication, 21-1, 22-1, 23-1
 - single database, 20-1
- Export utility, 11-8, 14-36, 16-35
- heterogeneous information sharing, 9-1
- high availability, 10-1
- Import utility, 11-8, 14-36, 16-35
- initialization parameters, 11-4, 19-4, 20-4, 21-3, 22-5
- instantiation, 11-8, 14-36
- JMS, 3-12
 - example, 19-35
- LOBs, 16-11
- logical change records (LCRs), 2-2
 - XML schema, A-1
- messaging, 13-18
- monitoring, 17-1
- network connectivity, 11-14
- OCI, 3-12
- overview, 1-2
- packages, 1-21
- point-in-time recovery, 16-29
- preparing for, 11-1
- propagation, 3-1
 - Oracle Real Application Clusters, 3-18
- rules, 6-1
 - action context, 6-18
 - evaluation context, 6-5, 6-15
 - event context, 6-18

- subset rules, 4-12, 6-5
- system-created, 6-3
- staging, 3-1
 - Oracle Real Application Clusters, 3-18
- Streams tool, 1-23
- supplemental logging, 2-11
- tags, 8-1
- trace files, 18-24
- transformations
 - rule-based, 6-25
- troubleshooting, 18-1
- OVERWRITE conflict resolution handler, 7-10

P

- PARALLEL_MAX_SERVERS initialization
 - parameter, 11-7
- point-in-time recovery
 - Oracle Streams, 16-29
- PREPARE_GLOBAL_INSTANTIATION
 - procedure, 2-13, 11-15, 12-11
- PREPARE_SCHEMA_INSTANTIATION
 - procedure, 2-13, 11-15, 12-11
- PREPARE_TABLE_INSTANTIATION
 - procedure, 2-13, 11-15, 12-11
- privileges
 - Oracle Streams administrator, 11-2
 - rules, 5-15
- PROCESSES initialization parameter, 11-7
- propagation jobs, 3-20
 - altering, 13-12
 - disabling, 13-17
 - enabling, 13-11
 - job queue processes, 3-20
 - managing, 13-7
 - monitoring, 17-15
 - RESTRICTED SESSION, 3-22
 - scheduling, 3-21, 13-11
 - trace files, 18-25
 - troubleshooting, 18-4
 - checking status, 18-6
 - job queue processes, 18-7
 - unscheduling, 13-13

- propagations, 3-1
 - architecture, 3-19
 - creating, 13-8
 - database links
 - creating, 21-7, 22-12
 - destination queue, 3-2
 - directed networks, 3-7
 - dropping, 13-18
 - ensured delivery, 3-6
 - managing, 13-7
 - monitoring, 17-15
 - queue buffers, 3-19
 - queues, 3-4
 - rule sets
 - removing, 13-16
 - specifying, 13-14
 - rules, 3-5, 6-2
 - adding, 13-14
 - removing, 13-16
 - source queue, 3-2
 - transformations
 - rule-based, 6-30
 - SYS.AnyData to typed queue, 13-24, 13-28
 - troubleshooting, 18-4
 - checking queues, 18-5
 - security, 18-8

Q

- queue buffers, 3-19
- queue forwarding, 3-8
- queues
 - AnyData, 3-11, 13-18
 - creating, 13-2
 - dequeuing, 19-25, 20-18
 - dropping, 13-7
 - enqueueing, 19-12, 20-16
 - user-defined types, 3-17
 - nontransactional, 3-24
 - propagation, 3-16
 - secure, 3-22
 - disabling user access, 13-5
 - enabling user access, 13-3
 - transactional, 3-24

R

RESNAME_ARRAY type, 24-14, 24-24, 24-34

RESNV_ARRAY type, 22-35

RESNV_LIST type, 5-12, 22-35, 24-6, 24-13, 24-14, 24-24, 24-31, 24-34

ADD_PAIR member procedure, 15-16, 15-20

REMOVE_PAIR member procedure, 15-20, 15-21

RESRULE_HIT_LIST type, 24-8, 24-14, 24-24, 24-34

RETABLE_ALIAS_LIST type, 24-12, 24-21, 24-31

RETABLE_VALUE type, 24-14, 24-24, 24-34

RETABLE_VALUE_LIST type, 24-14, 24-24, 24-34

RESVARIABLE_TYPE_LIST type, 24-5, 24-21, 24-31

RESVARIABLE_VALUE type, 24-8, 24-24

RESVARIABLE_VALUE_LIST type, 24-8, 24-24

redo logs

capture process, 2-2

re-enqueue

captured events, 20-1

REMOVE_PAIR member procedure, 15-20, 15-21

REMOVE_RULE procedure, 12-6, 13-16, 14-9, 15-7

replication

Oracle Streams, 21-1, 22-1, 23-1

adding databases, 22-76

adding objects, 22-64

resolution columns, 7-14

RESTRICTED SESSION system privilege

apply processes, 4-29

capture processes, 2-16

propagation jobs, 3-22

REVOKE_OBJECT_PRIVILEGE procedure, 5-15

REVOKE_SYSTEM_PRIVILEGE procedure, 5-15

row migration, 4-12

rule sets, 5-2

adding rules to, 15-5

creating, 15-2

dropping, 15-8

evaluation, 5-12

partial, 5-14

object privileges

granting, 15-9

revoking, 15-10

removing rules from, 15-7

system privileges

granting, 15-9

revoking, 15-10

rule-based transformations, 6-25

rules, 5-1

action contexts, 5-9

adding name-value pairs, 15-16, 15-20

removing name-value pairs, 15-20, 15-21

transformations, 6-25

ADD_RULE procedure, 5-8

altering, 15-5

apply process, 4-2, 6-2

capture process, 2-5, 6-2

components, 5-2

creating, 15-3

DBMS_RULE package, 5-12

dropping, 15-7

EVALUATE procedure, 5-12

evaluation, 5-12

capture process, 2-27

partial, 5-14

evaluation contexts, 5-5

creating, 19-19, 24-5, 24-12, 24-21, 24-31

evaluation function, 5-8

user-created, 6-24

variables, 5-6

event context, 5-12

example applications, 24-1

explicit variables, 5-6

example, 24-3, 24-18

implicit variables, 5-6

example, 24-27

managing, 15-2

maybe_rules, 5-12

monitoring, 17-42

object privileges

granting, 15-9

revoking, 15-10

partial evaluation, 5-14

privileges, 5-15

managing, 15-8

propagations, 3-5, 6-2

- rule conditions, 5-2, 6-6
 - complex, 6-20
 - explicit variables, 5-6
 - finding patterns in, 17-49
 - implicit variables, 5-6
 - types of operations, 6-22
 - undefined variables, 6-22
 - using NOT, 6-20
 - variables, 6-7
- rule_hits, 5-12
- simple rules, 5-3
- subset
 - querying for action context of, 15-15
 - querying for names of, 15-15
- system privileges
 - granting, 15-9
 - revoking, 15-10
- system-created, 6-1, 6-3
 - action context, 6-18
 - DDL rules, 6-9
 - DML rules, 6-7
 - evaluation context, 6-5, 6-15
 - event context, 6-18
 - global, 6-14
 - modifying, 15-6
 - schema, 6-12
 - STREAMSSEVALUATION_CONTEXT, 6-5, 6-15
 - subset rules, 4-12, 6-5, 6-6
 - table, 6-6
 - tags, 6-8, 8-3
- table data
 - example, 24-9, 24-18, 24-27
- troubleshooting, 18-17
- user-created, 6-19
- variables, 5-6

S

- SCHEDULE_PROPAGATION procedure, 13-11
- secure queues, 3-22
 - disabling user access, 13-5
 - enabling user access, 13-3
 - propagation, 18-8

- SESSIONS initialization parameter, 11-7
- SET_COMMAND_TYPE member procedure, 20-16
- SET_DML_HANDLER procedure, 4-6, 7-15
 - removing a DML handler, 14-18
 - removing an error handler, 14-27
 - setting a DML handler, 14-16
 - setting an error handler, 14-26
- SET_GLOBAL_INSTANTIATION_SCN procedure, 11-15, 14-35, 14-38
- SET_KEY_COLUMNS procedure, 4-11
 - removing substitute key columns, 14-29
 - setting substitute key columns, 14-28
- SET_OBJECT_NAME member procedure, 20-16, 22-34
- SET_PARAMETER procedure
 - apply process, 14-11, 18-12
 - capture process, 12-8
- SET_SCHEMA_INSTANTIATION_SCN procedure, 11-15, 14-35, 14-38
- SET_TABLE_INSTANTIATION_SCN procedure, 11-15, 14-35
- SET_TAG procedure, 8-2, 16-26
- SET_TRIGGER_FIRING_PROPERTY procedure, 4-25
- SET_UP_QUEUE procedure, 13-2
- SET_UPDATE_CONFLICT_HANDLER procedure, 7-8
 - modifying an update conflict handler, 14-31
 - removing an update conflict handler, 14-32
 - setting an update conflict handler, 14-30
- SET_VALUE member procedure
 - LCRs, 15-12
- SET_VALUES member procedure, 14-24, 20-16
- SGA_MAX_SIZE initialization parameter, 11-7
- SHARED_POOL_SIZE initialization parameter, 11-8
- SOAP
 - Streams queues, 3-16
- source queue, 3-2
- SQL*Loader
 - capture processes, 2-10
- staging, 3-1
 - architecture, 3-19
 - events, 3-3
 - heterogeneous environments, 9-3

- management, 13-1
- queue buffers, 3-19
- secure queues, 3-22
 - disabling user access, 13-5
 - enabling user access, 13-3
- start SCN, 2-15, 2-16
- START_APPLY procedure, 14-7
- START_CAPTURE procedure, 12-5
- STOP_APPLY procedure, 14-7
- STOP_CAPTURE procedure, 12-14
- Streams. *See* Oracle Streams
- Streams tool, 1-23
- STREAMS\$EVALUATION_CONTEXT, 6-5, 6-15
- STREAMS\$TRANSFORM_FUNCTION, 6-26
- STREAMS_CONFIGURATION parameter
 - for Import utility, 11-10
- STREAMS_INSTANTIATION parameter
 - for Import utility, 11-10, 21-15, 22-28, 23-49
- supplemental logging
 - capture process, 2-11
 - column lists, 7-12
 - DBA_LOG_GROUPS view, 17-11
 - example, 20-12, 21-12, 22-23, 23-26
 - instantiation, 2-15
 - row subsetting, 4-14
 - specifying, 12-9
- SYS.AnyData. *See Also* AnyData datatype
- system change numbers (SCN)
 - applied SCN for a capture process, 2-16, 17-7
 - applied SCN for an apply process, 4-28
 - captured SCN for a capture process, 2-16
 - oldest SCN for an apply process, 4-28
 - start SCN for a capture process, 2-15, 2-16
- system-generated names
 - apply process, 4-23

T

- tags, 8-1
 - ALTER_APPLY procedure, 8-2, 8-6
 - apply process, 8-6
 - change cycling
 - avoidance, 8-8
 - CREATE_APPLY procedure, 8-2, 8-6
 - examples, 8-8

- getting value for current session, 16-27
- managing, 16-26
- monitoring, 17-50
 - apply process value, 17-51
 - current session value, 17-50
- online backups, 8-5
- removing value for apply process, 16-28
- rules, 6-8, 8-3
 - include_tagged_lcr parameter, 8-3
- SET_TAG procedure, 8-2
- setting value for apply process, 16-28
- setting value for current session, 16-26
- trace files
 - Oracle Streams, 18-24
- transformations
 - heterogeneous environments
 - Oracle to non-Oracle, 9-9
 - Oracle Streams, 6-25
 - propagations, 13-24, 13-28
 - rule-based
 - action context, 6-25
 - altering, 15-18
 - apply errors, 6-34
 - apply process, 6-32
 - capture process, 6-28
 - creating, 15-11, 22-34
 - errors, 6-30, 6-32, 6-34
 - managing, 15-11
 - multiple, 6-34
 - propagations, 6-30
 - removing, 15-21
 - STREAMS\$TRANSFORM_FUNCTION, 6-26
 - troubleshooting, 18-23
- triggers
 - firing property, 4-25
 - system triggers
 - on SCHEMA, 4-26
- troubleshooting
 - apply process, 18-9
 - checking apply handlers, 18-12
 - checking event type, 18-11
 - checking status, 18-10
 - exception queue, 18-13

- capture process, 18-2
 - checking progress, 18-3
 - checking status, 18-2
 - log parallelism, 18-3
 - persistent sessions, 18-4
- Oracle Streams, 18-1
- propagation jobs, 18-4
 - checking status, 18-6
 - job queue processes, 18-7
- propagations, 18-4
 - checking queues, 18-5
 - security, 18-8
- rule-based transformations, 18-23
- rules, 18-17

U

UNRECOVERABLE

- capture process, 2-9

UNRECOVERABLE clause

- SQL*Loader

- capture process, 2-10

UNSCHEDULE_PROPAGATION

- procedure, 13-13

use_old parameter

- in row LCR member functions, 16-8

user-defined datatypes

- AnyData queues, 3-17

V

VSSESSION view, 17-4, 17-28, 17-29, 17-30, 17-33

VSSTREAMS_APPLY_COORDINATOR

- view, 17-30, 17-31

VSSTREAMS_APPLY_READER view, 17-28, 17-29

VSSTREAMS_APPLY_SERVER view, 17-33, 17-35

VSSTREAMS_CAPTURE view, 17-4, 17-7, 17-8,
18-3

X

XML Schema

- for LCRs, A-1