Oracle® Streams

Replication Administrator's Guide

11*g* Release 2 (11.2)

E10705-07

October 2010

ORACLE®

Oracle Streams Replication Administrator's Guide, 11*g* Release 2 (11.2)

E10705-07

Copyright © 2003, 2010, Oracle and/or its affiliates. All rights reserved.

Primary Author: Randy Urbano

Contributors: Nimar Arora, Lance Ashdown, Ram Avudaiappan, Neerja Bhatt, Ragamayi Bhyravabhotla, Alan Downing, Curt Elsbernd, Yong Feng, Jairaj Galagali, Lei Gao, Thuvan Hoang, Lewis Kaplan, Tianshu Li, Jing Liu, Edwina Lu, Raghu Mani, Rui Mao, Pat McElroy, Shailendra Mishra, Valarie Moore, Bhagat Nainani, Maria Pratt, Arvind Rajaram, Viv Schupmann, Vipul Shah, Neeraj Shodhan, Wayne Smith, Jim Stamos, Janet Stern, Mahesh Subramaniam, Bob Thome, Byron Wang, Wei Wang, James M. Wilson, Lik Wong, Jingwei Wu, Haobo Xu, Jun Yuan, David Zhang

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this software or related documentation is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications which may create a risk of personal injury. If you use this software in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of this software. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software in dangerous applications.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

This software and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Contents

Preface xiii

Audience xiii
Documentation Accessibility xiii
Related Documents xiv
Conventions xiv

Part I Configuring Oracle Streams Replication

1 Preparing for Oracle Streams Replication

Overview of Oracle Streams Replication 1-1
Common Reasons to Use Oracle Streams Replication 1-2
Rules in an Oracle Streams Replication Environment 1-3
Decisions to Make Before Configuring Oracle Streams Replication 1-4
Decide Which Type of Replication Environment to Configure 1-4
Decide Whether to Configure Local or Downstream Capture for the Source Database 1-5
Decide Whether Changes Are Allowed at One Database or at Multiple Databases 1-7
Decide Whether the Replication Environment Will Have Nonidentical Replicas 1-8
Decide Whether the Replication Environment Will Use Apply Handlers 1-9
Decide Whether to Maintain DDL Changes 1-10
Decide How to Configure the Replication Environment 1-10
Tasks to Complete Before Configuring Oracle Streams Replication 1-13
Configuring an Oracle Streams Administrator on All Databases 1-13
Configuring Network Connectivity and Database Links 1-17
Ensuring That Each Source Database Is In ARCHIVELOG Mode 1-19
Setting Initialization Parameters Relevant to Oracle Streams 1-19
Configuring the Oracle Streams Pool 1-25
Specifying Supplemental Logging 1-28
Configuring Log File Transfer to a Downstream Capture Database 1-33
Adding Standby Redo Logs for Real-Time Downstream Capture 1-36

2 Simple Oracle Streams Replication Configuration

Configuring Replication Using the Setup Streams Replication Wizard 2-1
Configuring Replication Using the DBMS_STREAMS_ADM Package 2-3
The Oracle Streams Replication Configuration Procedures 2-4
Important Considerations for the Configuration Procedures 2-7

Creating the Required Directory Objects 2-15
Examples That Configure Two-Database Replication with Local Capture 2-16
Examples That Configure Two-Database Replication with Downstream Capture 2-32
Example That Configures Hub-and-Spoke Replication 2-48
Monitoring Oracle Streams Configuration Progress 2-53

3 Flexible Oracle Streams Replication Configuration

Creating a New Oracle Streams Single-Source Environment 3-2
Creating a New Oracle Streams Multiple-Source Environment 3-6
Configuring Populated Databases When Creating a Multiple-Source Environment 3-9
Adding Replicated Objects to Import Databases When Creating a New Environment 3-10
Complete the Multiple-Source Environment Configuration 3-11

4 Adding to an Oracle Streams Replication Environment

About Adding to an Oracle Streams Replication Environment 4-1
About Using the Setup Streams Replication Wizard or a Single Configuration Procedure 4-2
About Adding the Oracle Streams Components Individually in Multiple Steps 4-3
Adding Multiple Components Using a Single Procedure 4-3
Adding Database Objects to a Replication Environment Using a Single Procedure 4-4
Adding a Database to a Replication Environment Using a Single Procedure 4-7
Adding Components Individually in Multiple Steps 4-10
Adding Replicated Objects to an Existing Single-Source Environment 4-11
Adding a New Destination Database to a Single-Source Environment 4-15
Adding Replicated Objects to an Existing Multiple-Source Environment 4-18
Adding a New Database to an Existing Multiple-Source Environment 4-25

5 Configuring Implicit Capture

Configuring a Capture Process 5-1
Preparing to Configure a Capture Process 5-2
Configuring a Local Capture Process 5-3
Configuring a Downstream Capture Process 5-7
After Configuring a Capture Process 5-17
Configuring Synchronous Capture 5-17
Preparing to Configure a Synchronous Capture 5-18
Configuring a Synchronous Capture Using the DBMS_STREAMS_ADM Package 5-18
Configuring a Synchronous Capture Using the DBMS_CAPTURE_ADM Package 5-20
After Configuring a Synchronous Capture 5-22

6 Configuring Queues and Propagations

Creating an ANYDATA Queue 6-1
Creating Oracle Streams Propagations Between ANYDATA Queues 6-3
Preparing to Create a Propagation 6-4
Creating a Propagation Using DBMS_STREAMS_ADM 6-4
Creating a Propagation Using DBMS_PROPAGATION_ADM 6-6

7 Configuring Implicit Apply

Overview of Apply Process Creation 7-1
Preparing to Create an Apply Process 7-2
Creating an Apply Process for Captured LCRs Using DBMS_STREAMS_ADM 7-3
Creating an Apply Process Using DBMS_APPLY_ADM 7-4
Creating an Apply Process for Captured LCRs with DBMS_APPLY_ADM 7-4
Creating an Apply Process for Persistent LCRs with DBMS_APPLY_ADM 7-6

8 Instantiation and Oracle Streams Replication

Overview of Instantiation and Oracle Streams Replication 8-1
Capture Rules and Preparation for Instantiation 8-3
DBMS_STREAMS_ADM Package Procedures Automatically Prepare Objects 8-4
When Preparing for Instantiation Is Required 8-5
Supplemental Logging Options During Preparation for Instantiation 8-6
Preparing Database Objects for Instantiation at a Source Database 8-8
Aborting Preparation for Instantiation at a Source Database 8-13
Oracle Data Pump and Oracle Streams Instantiation 8-13
Data Pump Export and Object Consistency 8-13
Oracle Data Pump Import and Oracle Streams Instantiation 8-14
Instantiating Objects Using Data Pump Export/Import 8-17
Recovery Manager (RMAN) and Oracle Streams Instantiation 8-19
Instantiating Objects in a Tablespace Using Transportable Tablespace or RMAN 8-20
Instantiating an Entire Database Using RMAN 8-25
Setting Instantiation SCNs at a Destination Database 8-36
Setting Instantiation SCNs Using Export/Import 8-37
Setting Instantiation SCNs Using the DBMS_APPLY_ADM Package 8-38
Monitoring Instantiation 8-41
Determining Which Database Objects Are Prepared for Instantiation 8-41
Determining the Tables for Which an Instantiation SCN Has Been Set 8-42

9 Oracle Streams Conflict Resolution

About DML Conflicts in an Oracle Streams Environment 9-1
Conflict Types in an Oracle Streams Environment 9-2
Update Conflicts in an Oracle Streams Environment 9-2
Uniqueness Conflicts in an Oracle Streams Environment 9-2
Delete Conflicts in an Oracle Streams Environment 9-2
Foreign Key Conflicts in an Oracle Streams Environment 9-2
Conflicts and Transaction Ordering in an Oracle Streams Environment 9-3
Conflict Detection in an Oracle Streams Environment 9-3
Control Over Conflict Detection for Nonkey Columns 9-4
Rows Identification During Conflict Detection in an Oracle Streams Environment 9-4
Conflict Avoidance in an Oracle Streams Environment 9-4
Use a Primary Database Ownership Model 9-4
Avoid Specific Types of Conflicts 9-5
Conflict Resolution in an Oracle Streams Environment 9-6
Prebuilt Update Conflict Handlers 9-6

Custom Conflict Handlers 9-11
Managing Oracle Streams Conflict Detection and Resolution 9-12
Setting an Update Conflict Handler 9-12
Modifying an Existing Update Conflict Handler 9-13
Removing an Existing Update Conflict Handler 9-14
Stopping Conflict Detection for Nonkey Columns 9-14
Monitoring Conflict Detection and Update Conflict Handlers 9-16
Displaying Information About Conflict Detection 9-16
Displaying Information About Update Conflict Handlers 9-17

10 Oracle Streams Tags

Introduction to Tags 10-1
Tags and Rules Created by the DBMS_STREAMS_ADM Package 10-2
Tags and Online Backup Statements 10-4
Tags and an Apply Process 10-5
Oracle Streams Tags in a Replication Environment 10-6
N-Way Replication Environments 10-6
Hub-and-Spoke Replication Environments 10-9
Hub-and-Spoke Replication Environment with Several Extended Secondary Databases. 10-14
Managing Oracle Streams Tags 10-16
Managing Oracle Streams Tags for the Current Session 10-16
Managing Oracle Streams Tags for an Apply Process 10-17
Monitoring Oracle Streams Tags 10-18
Displaying the Tag Value for the Current Session 10-18
Displaying the Default Tag Value for Each Apply Process 10-19

11 Oracle Streams Heterogeneous Information Sharing

Oracle to Non-Oracle Data Sharing with Oracle Streams 11-1
Change Capture and Staging in an Oracle to Non-Oracle Environment 11-2
Change Apply in an Oracle to Non-Oracle Environment 11-3
Transformations in an Oracle to Non-Oracle Environment 11-7
Messaging Gateway and Oracle Streams 11-7
Error Handling in an Oracle to Non-Oracle Environment 11-8
Example Oracle to Non-Oracle Streams Environment 11-8
Non-Oracle to Oracle Data Sharing with Oracle Streams 11-8
Change Capture in a Non-Oracle to Oracle Environment 11-9
Staging in a Non-Oracle to Oracle Environment 11-9
Change Apply in a Non-Oracle to Oracle Environment 11-9
Instantiation from a Non-Oracle Database to an Oracle Database 11-10
Non-Oracle to Non-Oracle Data Sharing with Oracle Streams 11-10

Part II Administering Oracle Streams Replication

12 Managing Oracle Streams Replication

About Managing Oracle Streams 12-1
Tracking LCRs Through a Stream 12-1

Splitting and Merging an Oracle Streams Destination 12-5
About Splitting and Merging Oracle Streams 12-5
Split and Merge Options 12-12
Examples That Split and Merge Oracle Streams 12-14
Changing the DBID or Global Name of a Source Database 12-23
Resynchronizing a Source Database in a Multiple-Source Environment 12-24
Performing Database Point-in-Time Recovery in an Oracle Streams Environment 12-25
Performing Point-in-Time Recovery on the Source in a Single-Source Environment 12-25
Performing Point-in-Time Recovery in a Multiple-Source Environment 12-29
Performing Point-in-Time Recovery on a Destination Database 12-30
Running Flashback Queries in an Oracle Streams Replication Environment 12-34
Recovering from Operation Errors 12-36
Recovery Scenario 12-38

13 Comparing and Converging Data

About Comparing and Converging Data 13-1
Scans 13-2
Buckets 13-2
Parent Scans and Root Scans 13-3
How Scans and Buckets Identify Differences 13-3
Other Documentation About the DBMS_COMPARISON Package 13-6
Preparing To Compare and Converge a Shared Database Object 13-6
Diverging a Database Object at Two Databases to Complete Examples 13-7
Comparing a Shared Database Object at Two Databases 13-7
Comparing a Subset of Columns in a Shared Database Object 13-8
Comparing a Shared Database Object without Identifying Row Differences 13-9
Comparing a Random Portion of a Shared Database Object 13-11
Comparing a Shared Database Object Cyclically 13-12
Comparing a Custom Portion of a Shared Database Object 13-14
Comparing a Shared Database Object That Contains CLOB or BLOB Columns 13-16
Viewing Information About Comparisons and Comparison Results 13-19
Viewing General Information About the Comparisons in a Database 13-20
Viewing Information Specific to Random and Cyclic Comparisons 13-21
Viewing the Columns Compared by Each Comparison in a Database 13-22
Viewing General Information About Each Scan in a Database 13-24
Viewing the Parent Scan ID and Root Scan ID for Each Scan in a Database 13-26
Viewing Detailed Information About the Row Differences Found in a Scan 13-27
Viewing Information About the Rows Compared in Specific Scans 13-28
Converging a Shared Database Object 13-30
Converging a Shared Database Object for Consistency with the Local Object 13-31
Converging a Shared Database Object for Consistency with the Remote Object 13-32
Converging a Shared Database Object with a Session Tag Set 13-33
Rechecking the Comparison Results for a Comparison 13-34
Purging Comparison Results 13-35
Purging All of the Comparison Results for a Comparison 13-35
Purging the Comparison Results for a Specific Scan ID of a Comparison 13-36
Purging the Comparison Results of a Comparison Before a Specified Time 13-36

Dropping a Comparison 13-36
Using DBMS_COMPARISON in an Oracle Streams Replication Environment 13-37
Checking for Consistency After Instantiation 13-37
Checking for Consistency in a Running Oracle Streams Replication Environment 13-38

14 Managing Logical Change Records (LCRs)

Requirements for Managing LCRs 14-1
Constructing and Enqueuing LCRs 14-2
Executing LCRs 14-6
Executing Row LCRs 14-7
Executing DDL LCRs 14-10
Managing LCRs Containing LOB Columns 14-11
Apply Process Behavior for Direct Apply of LCRs Containing LOBs 14-12
LOB Assembly and Custom Apply of LCRs Containing LOB Columns 14-12
Requirements for Constructing and Processing LCRs Containing LOB Columns 14-18
Managing LCRs Containing LONG or LONG RAW Columns 14-21

Part III Oracle Streams Replication Best Practices

15 Best Practices for Oracle Streams Replication Databases

Best Practices for Oracle Streams Database Configuration 15-1
Use a Separate Queue for Capture and Apply Oracle Streams Clients 15-1
Automate the Oracle Streams Replication Configuration 15-2
Best Practices for Oracle Streams Database Operation 15-4
Follow the Best Practices for the Global Name of an Oracle Streams Database 15-4
Monitor Performance and Make Adjustments When Necessary 15-5
Monitor Capture Process's and Synchronous Capture's Queues for Size 15-5
Follow the Oracle Streams Best Practices for Backups 15-6
Adjust the Automatic Collection of Optimizer Statistics 15-7
Check the Alert Log for Oracle Streams Information 15-7
Follow the Best Practices for Removing an Oracle Streams Configuration at a Database 15-8
Best Practices for Oracle Real Application Clusters and Oracle Streams 15-8
Make Archive Log Files of All Threads Available to Capture Processes 15-8
Follow the Best Practices for the Global Name of an Oracle RAC Database 15-8
Follow the Best Practices for Configuring and Managing Propagations 15-9
Follow the Best Practices for Queue Ownership 15-9

16 Best Practices for Capture

Best Practices for Capture Process Configuration 16-1
Grant the Required Privileges to the Capture User 16-1
Set Capture Process Parallelism 16-2
Set the Checkpoint Retention Time 16-2
Best Practices for Capture Process Operation 16-3
Configure a Heartbeat Table at Each Source Database in an Oracle Streams Environment 16-3
Perform a Dictionary Build and Prepare Database Objects for Instantiation Periodically .. 16-4
Minimize the Performance Impact of Batch Processing 16-4

Best Practices for Synchronous Capture Configuration 16-4

17 Best Practices for Propagation

Best Practices for Propagation Configuration 17-1
Use Queue-to-Queue Propagations 17-1
Set the Propagation Latency for Each Propagation 17-2
Increase the SDU in a Wide Area Network for Better Network Performance 17-3
Best Practices for Propagation Operation 17-3
Restart Broken Propagations 17-3

18 Best Practices for Apply

Best Practices for Destination Database Configuration 18-1
Grant Required Privileges to the Apply User 18-1
Set Instantiation SCN Values 18-2
Configure Conflict Resolution 18-2
Best Practices for Apply Process Configuration 18-3
Set Apply Process Parallelism 18-3
Consider Allowing Apply Processes to Continue When They Encounter Errors 18-4
Best Practices for Apply Process Operation 18-4
Manage Apply Errors 18-4

Part IV Appendixes

A Migrating Advanced Replication to Oracle Streams

Overview of the Migration Process A-1
Migration Script Generation and Use A-2
Modification of the Migration Script A-2
Actions Performed by the Generated Script A-2
Migration Script Errors A-3
Manual Migration of Updatable Materialized Views A-3
Advanced Replication Elements that Cannot Be Migrated to Oracle Streams A-3
Preparing to Generate the Migration Script A-3
Generating and Modifying the Migration Script A-4
Example Advanced Replication Environment to be Migrated to Oracle Streams A-4
Performing the Migration for Advanced Replication to Oracle Streams A-8
Before Executing the Migration Script A-8
Executing the Migration Script A-10
After Executing the Script A-10
Re-creating Master Sites to Retain Materialized View Groups A-12

Index

List of Figures

1–1 Oracle Streams Information Flow 1-2
1–2 The Capture Database 1-6
2–1 Sample Oracle Streams Environment That Replicates an Entire Database 2-18
2–2 Two-Database Replication Environment with Local Capture Processes 2-25
2–3 Sample Oracle Streams Environment That Replicates Tables 2-29
2–4 Sample Oracle Streams Environment That Replicates Tablespaces 2-35
2–5 Two-Database Replication Environment with a Downstream Capture Process 2-39
2–6 Sample Oracle Streams Environment That Replicates a Schema 2-44
2–7 Hub-and-Spoke Environment with Capture Processes and Read/Write Spokes 2-50
3–1 Example Oracle Streams Single-Source Environment 3-3
3–2 Example Oracle Streams Multiple-Source Environment 3-7
4–1 Example of Adding Replicated Objects to a Single-Source Environment 4-12
4–2 Example of Adding a Destination to a Single-Source Environment 4-16
4–3 Example of Adding Replicated Objects to a Multiple-Source Environment 4-20
4–4 Example of Adding a Database to a Multiple-Source Environment 4-26
10–1 Each Database Is a Source and Destination Database 10-7
10–2 Tag Use When Each Database Is a Source and Destination Database 10-8
10–3 Primary Database Sharing Data with Several Secondary Databases 10-10
10–4 Tags Used at the Primary Database 10-13
10–5 Tags Used at a Secondary Database 10-14
10–6 Primary Database and Several Extended Secondary Databases 10-15
11–1 Oracle to Non-Oracle Heterogeneous Data Sharing 11-2
11–2 Non-Oracle to Oracle Heterogeneous Data Sharing 11-8
12–1 Problem Destination in an Oracle Streams Replication Environment 12-7
12–2 Splitting Oracle Streams 12-9
12–3 Cloned Stream Begins Flowing and Starts to Catch Up to One Original Stream 12-10
12–4 Merging Oracle Streams 12-11
13–1 Comparison with max_num_buckets=3 and Differences in Each Bucket of Each Scan 13-4
13–2 Comparison with max_num_buckets=3 and Differences in One Bucket of Each Scan. 13-5
A–1 Advanced Replication Environment to be Migrated to Oracle Streams A-4

List of Tables

1–1 Oracle Streams Replication Configuration Options 1-12
1–2 Initialization Parameters Relevant to Oracle Streams 1-20
2–1 Required Parameters for the Oracle Streams Replication Configuration Procedures..... 2-5
8–1 DBMS_CAPTURE_ADM Package Procedures That Are Run Automatically 8-4
8–2 Supplemental Logging Options During Preparation for Instantiation..... 8-7
8–3 Set Instantiation SCN Procedures and the Statements They Cover 8-39
9–1 Customized Sequences for Oracle Streams Replication Environments..... 9-5
14–1 LOB Data Type Representations in Row LCRs 14-11
14–2 Oracle Streams Behavior with LOB Assembly Disabled 14-13
14–3 Oracle Streams Behavior with LOB Assembly Enabled..... 14-13

Preface

Oracle Streams Replication Administrator's Guide describes the features and functionality of Oracle Streams that can be used for data replication. This document contains conceptual information about Oracle Streams replication, along with information about configuring and managing an Oracle Streams replication environment.

This Preface contains these topics:

- Audience
- Documentation Accessibility
- Related Documents
- Conventions

Audience

Oracle Streams Replication Administrator's Guide is intended for database administrators who create and maintain Oracle Streams replication environments. These administrators perform one or more of the following tasks

- Plan for an Oracle Streams replication environment
- Configure an Oracle Streams replication environment
- Configure conflict resolution in an Oracle Streams replication environment
- Administer an Oracle Streams replication environment
- Monitor an Oracle Streams replication environment
- Perform necessary troubleshooting activities for an Oracle Streams replication environment

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

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading

technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at http://www.oracle.com/accessibility/.

Accessibility of Code Examples in Documentation

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

Accessibility of Links to External Web Sites in Documentation

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

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/support/contact.html or visit http://www.oracle.com/accessibility/support.html if you are hearing impaired.

Related Documents

For more information, see these Oracle resources:

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

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

Conventions

The following text conventions are used in this document:

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

Part I

Configuring Oracle Streams Replication

This part describes configuring Oracle Streams replication and contains the following chapters:

- Chapter 1, "Preparing for Oracle Streams Replication"
- Chapter 2, "Simple Oracle Streams Replication Configuration"
- Chapter 3, "Flexible Oracle Streams Replication Configuration"
- Chapter 4, "Adding to an Oracle Streams Replication Environment"
- Chapter 5, "Configuring Implicit Capture"
- Chapter 6, "Configuring Queues and Propagations"
- Chapter 7, "Configuring Implicit Apply"
- Chapter 8, "Instantiation and Oracle Streams Replication"
- Chapter 9, "Oracle Streams Conflict Resolution"
- Chapter 10, "Oracle Streams Tags"
- Chapter 11, "Oracle Streams Heterogeneous Information Sharing"

1

Preparing for Oracle Streams Replication

This chapter contains information about preparing for an Oracle Streams replication environment. This chapter also describes best practices to follow when you are preparing for an Oracle Streams replication environment.

This chapter contains these topics:

- Overview of Oracle Streams Replication
- Decisions to Make Before Configuring Oracle Streams Replication
- Tasks to Complete Before Configuring Oracle Streams Replication

> **See Also:** *Oracle Streams Concepts and Administration* for general information about Oracle Streams. This document assumes that you understand the concepts described in *Oracle Streams Concepts and Administration*.

Overview of Oracle Streams Replication

Replication is the process of sharing database objects and data at multiple databases. To maintain replicated database objects and data at multiple databases, a change to one of these database objects at a database is shared with the other databases. Through this process, the database objects and data are kept synchronized at all of the databases in the replication environment. In an Oracle Streams replication environment, the database where a change originates is called the **source database**, and a database where a change is shared is called a **destination database**.

When you use Oracle Streams, replication of a data manipulation language (DML) or data definition language (DDL) change typically includes three steps:

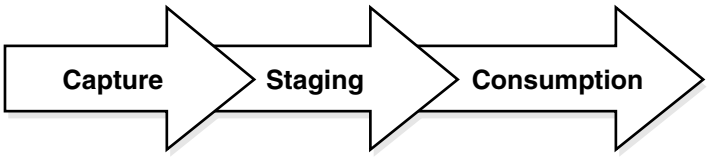
1. A capture process, a synchronous capture, or an application creates one or more logical change records (LCRs) and enqueues them. An LCR is a message with a specific format that describes a database change. A capture process reformats changes captured from the redo log into LCRs, a synchronous capture uses an internal mechanism to reformat changes into LCRs, and an application can construct LCRs. If the change was a DML operation, then each row LCR encapsulates a row change resulting from the DML operation to a replicated table at the source database. If the change was a DDL operation, then a DDL LCR encapsulates the DDL change that was made to a replicated database object at a source database.
2. A propagation propagates the staged LCRs to another queue, which usually resides in a database that is separate from the database where the LCRs were captured. An LCR can be propagated to several different queues before it arrives at a destination database.

3. At a destination database, an apply process consumes the change. An apply process can dequeue the LCR and apply it directly to the replicated database object, or an apply process can dequeue the LCR and send it to an apply handler. In an Oracle Streams replication environment, an apply handler performs customized processing of an LCR. An apply handler can apply the change in the LCR to the replicated database object, or it can consume the LCR in some other way.

Step 1 and Step 3 are required, but Step 2 is optional because, in some cases, a capture process or a synchronous capture can enqueue a change into a queue, and an apply process can dequeue the change from the same queue. An application can also enqueue an LCR directly at a destination database. In addition, in a heterogeneous replication environment in which an Oracle database shares information with a non-Oracle database, an apply process can apply changes directly to a non-Oracle database without propagating LCRs.

Figure 1–1 illustrates the information flow in an Oracle Streams replication environment.

Figure 1–1 Oracle Streams Information Flow



This document describes how to use Oracle Streams for replication and includes the following information:

- Conceptual information relating to Oracle Streams replication
- Instructions for configuring an Oracle Streams replication environment
- Instructions for administering, monitoring, and troubleshooting an Oracle Streams replication environment
- Examples that create and maintain Oracle Streams replication environments

Replication is one form of information sharing. Oracle Streams enables replication, and it also enables other forms of information sharing, such as messaging, event management and notification, data warehouse loading, and data protection.

> **See Also:** *Oracle Streams Concepts and Administration* for more information about Oracle Streams

Common Reasons to Use Oracle Streams Replication

The following are some of the most common reasons for using Oracle Streams replication:

- **Availability:** Replication provides fast, local access to shared data because it balances activity over multiple sites. Some users can access one server while other users access different servers, thereby reducing the load at all servers. Also, users can access data from the replication site that has the lowest access cost, which is typically the site that is geographically closest to them.
- **Performance and Network Load Reduction:** Replication provides fast, local access to shared data because it balances activity over multiple sites. Some users can access one server while other users access different servers, thereby reducing the

load at all servers. Applications can access various regional servers instead of accessing one central server. This configuration can reduce network load dramatically.

Rules in an Oracle Streams Replication Environment

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 Database. Rules control the information flow in an Oracle Streams replication environment. Each of the following components is a client of the rules engine:

- Capture process
- Synchronous capture
- Propagation
- Apply process

You control the behavior of each of these Oracle Streams clients using rules. A **rule set** contains a collection of rules. You can associate a positive and a negative rule set with a capture process, a propagation, and an apply process, but a synchronous capture can have only a positive rule set.

In a replication environment, an Oracle Streams client performs an action if a logical change record (LCR) satisfies its rule sets. In general, an LCR satisfies the rule sets for an Oracle Streams client if *no rules* in the negative rule set evaluate to `TRUE` for the LCR, and *at least one rule* in the positive rule set evaluates to `TRUE` for the LCR. If an Oracle Streams client is associated with both a positive and negative rule set, then the negative rule set is always evaluated first.

Specifically, you control the information flow in an Oracle Streams replication environment in the following ways:

- Specify the changes that a capture process captures from the redo log or discards. That is, if a change found in the redo log satisfies the rule sets for a capture process, then the capture process captures the change. If a change found in the redo log does not satisfy the rule sets for a capture process, then the capture process discards the change.
- Specify the changes that a synchronous capture captures or discards. That is, if a DML change made to a table satisfies the rule set for a synchronous capture, then the synchronous capture captures the change. If a DML change made to a table does not satisfy the rule set for a synchronous capture, then the synchronous capture discards the change.
- Specify the LCRs that a propagation propagates from one queue to another or discards. That is, if an LCR in a queue satisfies the rule sets for a propagation, then the propagation sends the LCR. If an LCR in a queue does not satisfy the rule sets for a propagation, then the propagation discards the LCR.
- Specify the LCRs that an apply process dequeues or discards. That is, if an LCR in a queue satisfies the rule sets for an apply process, then the apply process dequeues and processes the LCR. If an LCR in a queue does not satisfy the rule sets for an apply process, then the apply process discards the LCR.

You can use the Oracle-supplied PL/SQL package `DBMS_STREAMS_ADM` to create rules for an Oracle Streams replication environment. You can specify these system-created rules at the following levels:

- Table level - Contains a rule condition that evaluates to `TRUE` for changes made to a particular table
- Schema level - Contains a rule condition that evaluates to `TRUE` for changes made to a particular schema and the database objects in the schema
- Global level - Contains a rule condition that evaluates to `TRUE` for all changes made to a database

In addition, a single system-created rule can evaluate to `TRUE` for DML changes or for DDL changes, but not both. So, for example, to replicate both DML and DDL changes to a particular table, you need both a table-level DML rule and a table-level DDL rule for the table.

Oracle Streams also supports subsetting of table data with subset rules. If a replicated table in a database contains only a subset of the data, then you can configure Oracle Streams so that only the appropriate subset of the data is replicated. For example, a particular database might maintain data for employees in a particular department only. One or more other databases in the replication environment might contain all of the data in the employees table. In this case, you can use subset rules to replicate changes to the data for employees in that department with the subset table, but not changes to employees in other departments.

Subsetting can be done at any point in the Oracle Streams information flow. That is, a capture process or synchronous capture can use a subset rule to capture a subset of changes to a particular table, a propagation can use a subset rule to propagate a subset of changes to a particular table, and an apply process can use a subset rule to apply a subset of changes to a particular table.

Note: Synchronous captures only use table rules. Synchronous captures ignore schema and global rules.

See Also: *Oracle Streams Concepts and Administration* for more information about how rules are used in Oracle Streams

Decisions to Make Before Configuring Oracle Streams Replication

Make the following decisions before configuring Oracle Streams replication:

- Decide Which Type of Replication Environment to Configure
- Decide Whether to Configure Local or Downstream Capture for the Source Database
- Decide Whether Changes Are Allowed at One Database or at Multiple Databases
- Decide Whether the Replication Environment Will Have Nonidentical Replicas
- Decide Whether the Replication Environment Will Use Apply Handlers
- Decide Whether to Maintain DDL Changes
- Decide How to Configure the Replication Environment

Decide Which Type of Replication Environment to Configure

Before configuring a replication environment, first decide how many databases will be included in the replication environment, which database objects will be replicated, and how database changes will flow through the replication environment. Here are the most common types of replication environments:

- One-way replication in a two database environment where one database is read/write and the other database is read-only
- Bi-directional replication in a two database environment where both databases are read/write
- Hub-and-spoke replication with a read/write hub and read-only spokes
- Hub-and-spoke replication with a read/write hub and one or more read/write spokes
- N-way replication with multiple read/write databases

One of these environments meet the replication requirements of most organizations. *Oracle Database 2 Day + Data Replication and Integration Guide* describes these common types of replication environments in detail.

If these common replication environments do not meet your requirements, then you can configure almost any type of custom replication environment with Oracle Streams. For example, a custom replication environment might send database changes through several intermediary databases before the changes are applied at a destination database.

Decide Whether to Configure Local or Downstream Capture for the Source Database

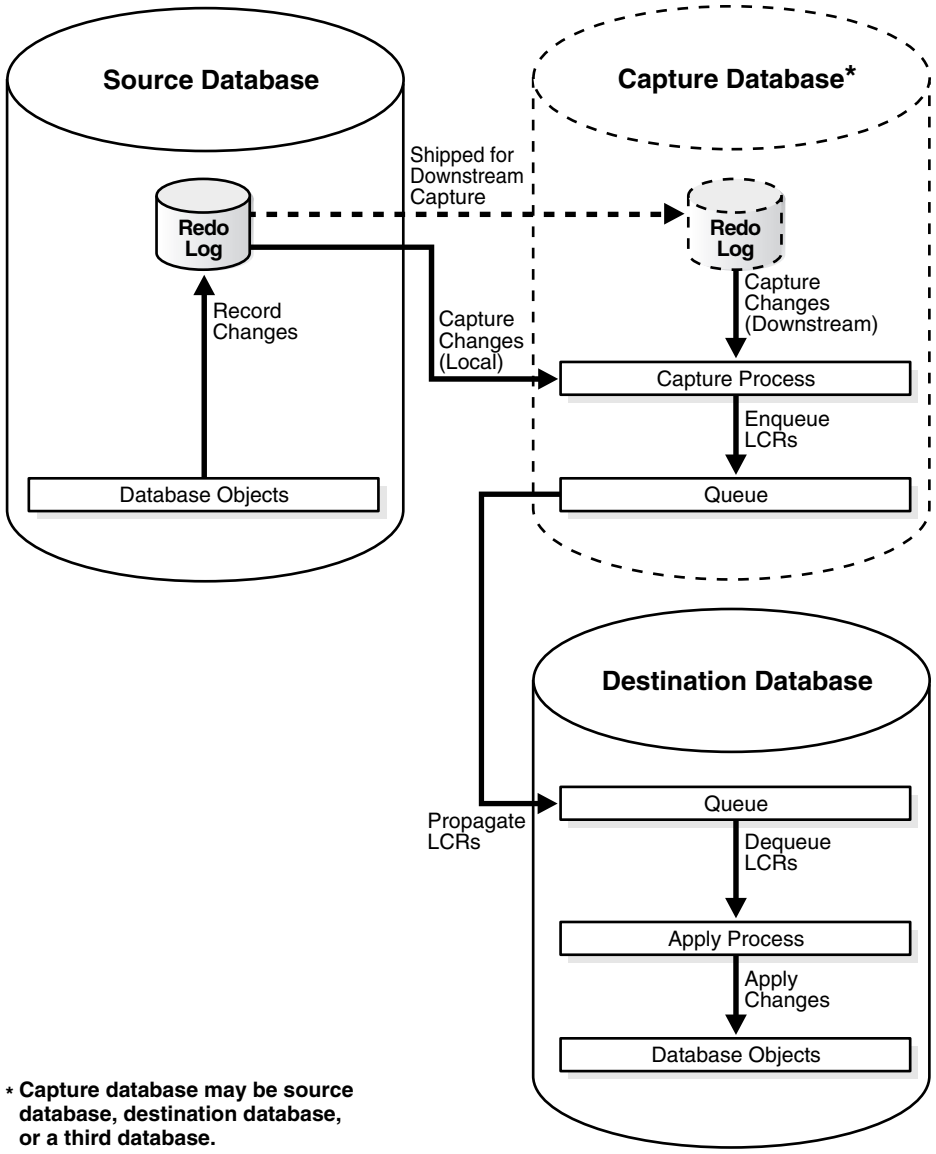
Local capture means that a capture process runs on the source database. Downstream capture means that a capture process runs on a database other than the source database. The primary reason to use downstream capture is to reduce the load on the source database, thereby improving its performance.

The database that captures changes made to the source database is called the **capture database**. One of the following databases can be the capture database:

- Source database (local capture)
- Destination database (downstream capture)
- A third database (downstream capture)

Figure 1–2 shows the role of the capture database.

Figure 1–2 The Capture Database



If the source database or a third database is the capture database, then a propagation sends changes from the capture database to the destination database. If the destination database is the capture database, then this propagation between databases is not needed because the capture process and apply process use the same queue.

If you decide to configure a downstream capture process, then you must decide which type of downstream capture process you want to configure. The following types are available:

- A **real-time downstream capture process** configuration means that redo transport services use the log writer process (LGWR) at the source database to send redo data to the downstream database, and a remote file server process (RFS) at the downstream database receives the redo data over the network and stores the redo data in the standby redo log.
- An **archived-log downstream capture process** configuration means that archived redo log files from the source database are copied to the downstream database, and the capture process captures changes in these archived redo log files. These

log files can be transferred automatically using redo transport services, or they can be transferred manually using a method such at FTP.

The advantage of real-time downstream capture over archived-log downstream capture is that real-time downstream capture reduces the amount of time required to capture changes made at the source database. The time is reduced because the real-time downstream capture process does not need to wait for the redo log file to be archived before it can capture changes from it. You can configure multiple real-time downstream capture processes that captures changes from the same source database, but you cannot configure real-time downstream capture for multiple source databases at one downstream database.

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

If you decide to configure a real-time downstream capture process, then you must complete the steps in "Configuring Log File Transfer to a Downstream Capture Database" on page 1-33 and "Adding Standby Redo Logs for Real-Time Downstream Capture" on page 1-36.

If you decide to configure an archived-log downstream capture process that uses archived redo log files that were transferred to the downstream database automatically by redo transport services, then you must complete the steps in "Configuring Log File Transfer to a Downstream Capture Database" on page 1-33.

Note: When the RMAN `DUPLICATE` or `CONVERT DATABASE` command is used for database instantiation with one of these procedures, the destination database cannot be the capture database.

See Also:

- *Oracle Streams Concepts and Administration* for information about local capture and downstream capture
- "Decide Whether Changes Are Allowed at One Database or at Multiple Databases" on page 1-7
- "Recovery Manager (RMAN) and Oracle Streams Instantiation" on page 8-19

Decide Whether Changes Are Allowed at One Database or at Multiple Databases

A replication environment can limit changes to a particular replicated database object to one database only. In this case, the replicated database object is read/write at one database and read-only at the other databases in the replication environment. Or, a replication environment can allow changes to a replicated database object at two or more databases.

When two or more databases can change a replicated database object, conflicts are possible. A conflict is a mismatch between the old values in an LCR and the expected data in a table. Conflicts can occur in an Oracle Streams replication environment that permits concurrent data manipulation language (DML) operations on the same data at multiple databases. Conflicts typically result when two or more databases make

changes to the same row in a replicated table at nearly the same time. If conflicts are not resolved, then they can result in inconsistent data at replica databases.

Typically, conflicts are possible in the following common types of replication environments:

- Bi-directional replication in a two database environment where the replicated database objects at both databases are read/write
- Hub-and-spoke replication where the replicated database objects are read/write at the hub and at one or more spokes
- N-way replication where the replicated database objects are read/write at multiple databases

Oracle Database 2 Day + Data Replication and Integration Guide describes these common types of replication environments in more detail.

Oracle Streams provides prebuilt conflict handlers to resolve conflicts automatically. You can also build your own custom conflict handler to resolve data conflicts specific to your business rules. Such a conflict handler can be part of a procedure DML handler or an error handler.

If conflicts are possible in the replication environment you plan to configure, then plan to create conflict handlers to resolve these conflicts.

See Also:

- "Configuring Network Connectivity and Database Links" on page 1-17
- "Ensuring That Each Source Database Is In ARCHIVELOG Mode" on page 1-19
- Chapter 9, "Oracle Streams Conflict Resolution"
- "Decide Whether to Configure Local or Downstream Capture for the Source Database" on page 1-5

Decide Whether the Replication Environment Will Have Nonidentical Replicas

Oracle Streams replication supports sharing database objects that are not identical at multiple databases. Different databases in the Oracle Streams environment can contain replicated database objects with different structures. In Oracle Streams replication, a **rule-based transformation** is any modification to a logical change record (LCR) that results when a rule in a positive rule set evaluates to `TRUE`. You can configure rule-based transformations during capture, propagation, or apply to make any necessary changes to LCRs so that they can be applied at a destination database.

For example, a table at a source database can have the same data as a table at a destination database, but some column names can be different. In this case, a rule-based transformation can change the names of the columns in LCRs from the source database so that they can be applied successfully at the destination database.

There are two types of rule-based transformations: declarative and custom. **Declarative rule-based transformations** cover a set of common transformation scenarios for row LCRs, including renaming a schema, renaming a table, adding a column, renaming a column, keeping a list of columns, and deleting a column. You specify such a transformation using a procedure in the `DBMS_STREAMS_ADM` package. Oracle Streams performs declarative transformations internally, without invoking PL/SQL.

A **custom rule-based transformation** requires a user-defined PL/SQL function to perform the transformation. Oracle Streams invokes the PL/SQL function to perform the transformation. A custom rule-based transformation can modify captured LCRs, persistent LCRs, or user messages. For example, a custom rule-based transformation can change the data type of a particular column in an LCR. A custom rule-based transformation must be defined as a PL/SQL function that takes an `ANYDATA` object as input and returns an `ANYDATA` object.

Rule-based transformations can be done at any point in the Oracle Streams information flow. That is, a capture process or a synchronous capture can perform a rule-based transformation on a change when a rule in its positive rule set evaluates to `TRUE` for the change. Similarly, a propagation or an apply process can perform a rule-based transformation on an LCR when a rule in its positive rule set evaluates to `TRUE` for the LCR.

If you plan to have nonidentical copies of database objects in your replication environment, then plan to create rule-based transformations that will modify LCRs so that they can be applied successfully at destination databases.

Note: Throughout this document, "rule-based transformation" is used when the text applies to both declarative and custom rule-based transformations. This document distinguishes between the two types of rule-based transformations when necessary.

See Also: *Oracle Streams Concepts and Administration* for more information about rule-based transformations

Decide Whether the Replication Environment Will Use Apply Handlers

When you use an apply handler, an apply process passes a message to either a collection of SQL statements or a user-created PL/SQL procedure for processing.

The following types of apply handlers are possible:

- A statement DML handler uses a collection of SQL statement to process row logical change records (row LCRs).
- A procedure DML handler uses a PL/SQL procedure to process row LCRs.
- A DDL handler uses a PL/SQL procedure to process DDL LCRs.
- A message handler uses a PL/SQL procedure to process user messages.
- A precommit handlers uses a PL/SQL procedure to process the commit information for a transaction.
- An error handler uses a PL/SQL procedure to process row LCRs that have caused apply errors.

An apply handler can process a message in a customized way. For example, a handler might audit the changes made to a table or enqueue an LCR into a queue after the change in the LCR has been applied. An application can then process the re-enqueued LCR. A handler might also be used to audit the changes made to a database.

If you must process LCRs in a customized way in your replication environment, then decide which apply handlers you should use to accomplish your goals. Next, create the PL/SQL procedures that will perform the custom processing and specify these procedures as apply handlers when your environment is configured.

See Also: *Oracle Streams Concepts and Administration*

Decide Whether to Maintain DDL Changes

Replication environments typically maintain data manipulation language (DML) changes to the replicated database objects. DML changes include `INSERT`, `UPDATE`, `DELETE`, and LOB update operations. You must decide whether you want the replication environment to maintain data definition language (DDL) changes as well. Examples of statements that result in DDL changes are `CREATE TABLE`, `ALTER TABLE`, `ALTER TABLESPACE`, and `ALTER DATABASE`.

Some Oracle Streams replication environments assume that the database objects are the same at each database. In this case, maintaining DDL changes with Oracle Streams makes it easy to keep the shared database objects synchronized. However, some Oracle Streams replication environments require that shared database objects are different at different databases. For example, a table can have a different name or shape at two different databases. In these environments, rule-based transformations and apply handlers can modify changes so that they can be shared between databases, and you might not want to maintain DDL changes with Oracle Streams. In this case, you should make DDL changes manually at each database that required them.

When replicating data definition language (DDL) changes, do not allow system-generated names for constraints or indexes. Modifications to these database objects will most likely fail at the destination database because the object names at the different databases will not match. Also, storage clauses might cause problems if the destination databases are not identical. If you decide not to replicate DDL in your Oracle Streams environment, then any table structure changes must be performed manually at each database in the environment.

See Also:

- "Data Definition Language (DDL) Changes" on page 2-13
- "Decide Whether the Replication Environment Will Have Nonidentical Replicas" on page 1-8
- *Oracle Streams Concepts and Administration* for more information about rule-based transformations

Decide How to Configure the Replication Environment

There are three options for configuring an Oracle Streams replication environment:

- Run the Setup Streams Replication wizard to configure replication between two databases. You can run the wizard multiple times to configure a replication environment with more than two databases.

  The wizard walks you through the process of configuring your replication environment, but there are some limits to the types of replication environments that can be configured with the wizard. For example, the wizard currently cannot configure synchronous capture.

  See "Configuring Replication Using the Setup Streams Replication Wizard" on page 2-1, *Oracle Database 2 Day + Data Replication and Integration Guide*, and the Oracle Enterprise Manager online help for more information about the replication configuration wizards.

- Run a configuration procedure in the `DBMS_STREAMS_ADM` supplied PL/SQL package to configure replication between two databases. You can run the procedure multiple times to configure a replication environment with more than two databases.

The following procedures configure Oracle Streams replication:

- The `MAINTAIN_GLOBAL` procedure configures an Oracle Streams environment that replicates changes at the database level between two databases.
- The `MAINTAIN_SCHEMAS` procedure configures an Oracle Streams environment that replicates changes to specified schemas between two databases.
- The `MAINTAIN_SIMPLE_TTS` procedure clones a simple tablespace from a source database at a destination database and uses Oracle Streams to maintain this tablespace at both databases.
- The `MAINTAIN_TABLES` procedure configures an Oracle Streams environment that replicates changes to specified tables between two databases.
- The `MAINTAIN_TTS` procedure clones a set of tablespaces from a source database at a destination database and uses Oracle Streams to maintain these tablespaces at both databases.

These procedures configure multiple Oracle Streams components with a single procedure call, and they automatically follow Oracle Streams best practices. They are ideal for configuring one-way, bi-directional, and hub-and-spoke replication environments.

See "Configuring Replication Using the DBMS_STREAMS_ADM Package" on page 2-3 and *Oracle Database PL/SQL Packages and Types Reference* for more information about these procedures.

- Configure each Oracle Streams component separately. These components include queues, capture processes, synchronous captures, propagations, and apply processes. Choose this option if you plan to configure an n-way replication environment, or if you plan to configure another type of replication environment that cannot be configured with the wizards or configuration procedures.

  See Chapter 3, "Flexible Oracle Streams Replication Configuration" for information about configuring each component of a replication environment separately.

Your configuration options might be limited by the type of replication environment you want to configure. See "Decide Which Type of Replication Environment to Configure" on page 1-4.

Table 1–1 lists the configuration options that are available for each type of replication environment.

Table 1–1 Oracle Streams Replication Configuration Options

Type of Replication Environment	Configuration Options and Examples
One-way replication in a two database replication environment	Setup Streams Replication Wizard in Oracle Enterprise Manager. Examples: ▪ "Tutorial: Configuring Two-Database Replication with Local Capture Processes" in *Oracle Database 2 Day + Data Replication and Integration Guide* ▪ "Tutorial: Configuring Two-Database Replication with a Downstream Capture Process" in *Oracle Database 2 Day + Data Replication and Integration Guide* A configuration procedure in the `DBMS_STREAMS_ADM` supplied PL/SQL package. Examples: ▪ "Configuring Two-Database Schema Replication with Local Capture" on page 2-24 ▪ "Configuring Two-Database Table Replication with Local Capture" on page 2-27 ▪ "Configuring Tablespace Replication with Downstream Capture at Destination" on page 2-33 ▪ "Configuring Schema Replication with Downstream Capture at Destination" on page 2-38 Configure each Oracle Streams component individually. Examples: ▪ *Oracle Streams Extended Examples*
Bi-directional replication in a two database replication environment	Setup Streams Replication Wizard in Oracle Enterprise Manager. Example: ▪ "Tutorial: Configuring Two-Database Replication with Local Capture Processes" in *Oracle Database 2 Day + Data Replication and Integration Guide* A configuration procedure in the `DBMS_STREAMS_ADM` supplied PL/SQL package. Examples: ▪ "Configuring Two-Database Global Replication with Local Capture" on page 2-16 ▪ "Configuring Two-Database Schema Replication with Local Capture" on page 2-24 ▪ "Configuring Schema Replication with Downstream Capture at Third Database" on page 2-42 Configure each Oracle Streams component individually. Example: ▪ "Tutorial: Configuring Two-Database Replication with Synchronous Captures" in *Oracle Database 2 Day + Data Replication and Integration Guide*
Hub-and-spoke replication with a read/write hub and read-only spokes	A configuration procedure in the `DBMS_STREAMS_ADM` supplied PL/SQL package. Configure each Oracle Streams component individually.

Table 1–1 (Cont.) Oracle Streams Replication Configuration Options

Type of Replication Environment	Configuration Options and Examples
Hub-and-spoke replication with a read/write hub and one or more read/write spokes	Setup Streams Replication Wizard in Oracle Enterprise Manager. Example: ■ "Tutorial: Configuring Hub-and-Spoke Replication with Local Capture Processes" in *Oracle Database 2 Day + Data Replication and Integration Guide* A configuration procedure in the `DBMS_STREAMS_ADM` supplied PL/SQL package. Example: ■ "Example That Configures Hub-and-Spoke Replication" on page 2-48 Configure each Oracle Streams component individually.
N-way replication with multiple read/write databases	Configure each Oracle Streams component individually. Example: ■ *Oracle Streams Extended Examples*
Custom replication environment	Configure each Oracle Streams component individually. See Chapter 3, "Flexible Oracle Streams Replication Configuration" for instructions. Examples: ■ *Oracle Streams Extended Examples*

Before configuring the replication environment, complete the tasks in "Tasks to Complete Before Configuring Oracle Streams Replication".

Tasks to Complete Before Configuring Oracle Streams Replication

The following sections describe tasks to complete before configuring Oracle Streams replication:

- Configuring an Oracle Streams Administrator on All Databases
- Configuring Network Connectivity and Database Links
- Ensuring That Each Source Database Is In ARCHIVELOG Mode
- Setting Initialization Parameters Relevant to Oracle Streams
- Configuring the Oracle Streams Pool
- Specifying Supplemental Logging
- Configuring Log File Transfer to a Downstream Capture Database
- Adding Standby Redo Logs for Real-Time Downstream Capture

Configuring an Oracle Streams Administrator on All Databases

To configure and manage an Oracle 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 an Oracle Streams administrator, and the Oracle Streams administrator should not use the `SYSTEM` tablespace as its default tablespace.

Typically, the user name for the Oracle Streams administrator is `strmadmin`, but any user with the proper privileges can be an Oracle Streams administrator. The examples in this section use `strmadmin` for the Oracle Streams administrator user name.

Create a separate tablespace for the Oracle Streams administrator at each participating Oracle Streams database. This tablespace stores any objects created in the Oracle Streams administrator schema, including any spillover of messages from the buffered queues owned by the schema.

See Also: *Oracle Database 2 Day + Data Replication and Integration Guide* for instructions about creating an Oracle Streams administrator using Oracle Enterprise Manager

Complete the following steps to configure an Oracle Streams administrator at each database in the environment that will use Oracle Streams:

1. In SQL*Plus, connect as an administrative user who can create users, grant privileges, and create tablespaces. Remain connected as this administrative user for all subsequent steps.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

2. Either create a tablespace for the Oracle Streams administrator or use an existing tablespace. For example, the following statement creates a new tablespace for the Oracle Streams administrator:

```
CREATE TABLESPACE streams_tbs DATAFILE '/usr/oracle/dbs/streams_tbs.dbf'
  SIZE 25M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;
```

3. Create a new user to act as the Oracle Streams administrator or use an existing user. For example, to create a user named `strmadmin` and specify that this user uses the `streams_tbs` tablespace, run the following statement:

```
CREATE USER strmadmin IDENTIFIED BY password
   DEFAULT TABLESPACE streams_tbs
   QUOTA UNLIMITED ON streams_tbs;
```

   Note: Enter an appropriate password for the administrative user.

   See Also: *Oracle Database Security Guide* for guidelines for choosing passwords

4. Grant the Oracle Streams administrator `DBA` role:

```
GRANT DBA TO strmadmin;
```

   Note: The `DBA` role is required for a user to create or alter capture processes, synchronous captures, and apply processes. When the user does not need to perform these tasks, `DBA` role can be revoked from the user.

5. Run the `GRANT_ADMIN_PRIVILEGE` procedure in the `DBMS_STREAMS_AUTH` package.

   A user must have explicit `EXECUTE` privilege on a package to execute a subprogram in the package inside of a user-created subprogram, and a user must have explicit `SELECT` privilege on a data dictionary view to query the view inside of a user-created subprogram. These privileges cannot be through a role. You can run the `GRANT_ADMIN_PRIVILEGE` procedure to grant such privileges to the Oracle Streams administrator, or you can grant them directly.

Depending on the parameter settings for the `GRANT_ADMIN_PRIVILEGE` procedure, it either grants the privileges for an Oracle Streams administrator directly, or it generates a script that you can edit and then run to grant these privileges.

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

Use the GRANT_ADMIN_PRIVILEGE procedure to grant privileges directly:

Run the following procedure:

```
BEGIN
  DBMS_STREAMS_AUTH.GRANT_ADMIN_PRIVILEGE(
    grantee          => 'strmadmin',
    grant_privileges => TRUE);
END;
/
```

Use the GRANT_ADMIN_PRIVILEGE procedure to generate a script:

Complete the following steps:

a. Use the SQL statement `CREATE DIRECTORY` to create a directory object for the directory into which you want to generate the script. A directory object is similar to an alias for the directory. For example, to create a directory object called `strms_dir` for the `/usr/admin` directory on your computer system, run the following procedure:

```
CREATE DIRECTORY strms_dir AS '/usr/admin';
```

b. Run the `GRANT_ADMIN_PRIVILEGE` procedure to generate a script named `grant_strms_privs.sql` and place this script in the `/usr/admin` directory on your computer system:

```
BEGIN
  DBMS_STREAMS_AUTH.GRANT_ADMIN_PRIVILEGE(
    grantee          => 'strmadmin',
    grant_privileges => FALSE,
    file_name        => 'grant_strms_privs.sql',
    directory_name   => 'strms_dir');
END;
/
```

Notice that the `grant_privileges` parameter is set to `FALSE` so that the procedure does not grant the privileges directly. Also, notice that the directory object created in Step a is specified for the `directory_name` parameter.

c. Edit the generated script if necessary and save your changes.

d. Execute the script in SQL*Plus:

```
SET ECHO ON
SPOOL grant_strms_privs.out
@/usr/admin/grant_strms_privs.sql
SPOOL OFF
```

e. Check the spool file to ensure that all of the grants executed successfully. If there are errors, then edit the script to correct the errors and rerun it.

6. If necessary, grant the following additional privileges:

- If you plan to use Oracle Enterprise Manager to manage databases with Oracle Streams components, then configure the Oracle Streams administrator to be a Database Control administrator. Doing so grants additional privileges required by Oracle Enterprise Manager, such as the privileges required to run Oracle Enterprise Manager jobs. See *Oracle Database 2 Day DBA* for instructions.
- Grant the privileges for a remote Oracle Streams administrator to perform actions in the local database. Grant these privileges using the `GRANT_REMOTE_ADMIN_ACCESS` procedure in the `DBMS_STREAMS_AUTH` package. Grant this privilege if a remote Oracle Streams administrator will use a database link that connects to the local Oracle Streams administrator to perform administrative actions. Specifically, grant these privileges if either of the following conditions are true:
  - You plan to configure a downstream capture process at a remote downstream database that captures changes originating at the local source database, and the downstream capture process will use a database link to perform administrative actions at the source database.
  - You plan to configure an apply process at the local database and use a remote Oracle Streams administrator to set the instantiation SCN values for replicated database objects at the local database.
- If no apply user is specified for an apply process, then grant the Oracle Streams administrator the necessary privileges to perform DML and DDL changes on the apply objects owned by other users. If an apply user is specified, then the apply user must have these privileges. These privileges can be granted directly or through a role.
- If no apply user is specified for an apply process, then grant the Oracle Streams administrator `EXECUTE` privilege on any PL/SQL subprogram owned by another user that is executed by an Oracle Streams apply process. These subprograms can be used in apply handlers or error handlers. If an apply user is specified, then the apply user must have these privileges. These privileges must be granted directly. They cannot be granted through a role.
- Grant the Oracle Streams administrator `EXECUTE` privilege on any PL/SQL function owned by another user that is specified in a custom rule-based transformation for a rule used by an Oracle Streams capture process, synchronous capture, propagation, apply process, or messaging client. For a capture process or synchronous capture, if a capture user is specified, then the capture user must have these privileges. For an apply process, if an apply user is specified, then the apply user must have these privileges. These privileges must be granted directly. They cannot be granted through a role.
- Grant the Oracle Streams administrator privileges to alter database objects where appropriate. For example, if the Oracle Streams administrator must create a supplemental log group for a table in another schema, then the Oracle Streams administrator must have the necessary privileges to alter the table. These privileges can be granted directly or through a role.
- If the Oracle Streams administrator does not own the queue used by an Oracle Streams capture process, synchronous capture, propagation, apply process, or messaging client, and is not specified as the queue user for the queue when the queue is created, then the Oracle Streams administrator must be configured as a secure queue user of the queue if you want the Oracle Streams administrator to be able to enqueue messages into or dequeue messages from the queue. The Oracle Streams administrator might also need `ENQUEUE` or

DEQUEUE privileges on the queue, or both. See *Oracle Streams Concepts and Administration* for information about managing queues.

- Grant the Oracle Streams administrator EXECUTE privilege on any object types that the Oracle Streams administrator might need to access. These privileges can be granted directly or through a role.
- If the Oracle Streams administrator will use Data Pump to perform export and import operations on database objects in other schemas during an Oracle Streams instantiation, then grant the EXP_FULL_DATABASE and IMP_FULL_DATABASE roles to the Oracle Streams administrator.
- If Oracle Database Vault is installed, then the user who performs the following actions must be granted the BECOME USER system privilege:
  - Creates or alters a capture process
  - Creates or alters an apply process

  Granting the BECOME USER system privilege to the user who performs these actions is not required if Oracle Database Vault is not installed. You can revoke the BECOME USER system privilege from the user after the completing one of these actions, if necessary.

7. Repeat all of the previous steps at each database in the environment that will use Oracle Streams.

See Also:

- "Grant the Required Privileges to the Capture User" on page 16-1
- "Grant Required Privileges to the Apply User" on page 18-1

Configuring Network Connectivity and Database Links

If you plan to use Oracle 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.
- For non-Oracle databases, configure an Oracle Database Gateway for communication between the Oracle database and the non-Oracle database.
- If you plan to propagate messages from a source queue at a database to a destination queue at another database, then create a private database link between the database containing the source queue and the database containing the destination queue. Each database link should use a CONNECT TO clause for the user propagating messages between databases.

A database link from the source database to the destination database is always required. The name of the database link must match the global name of the destination database.

A database link from the destination database to the source database is required in any of the following cases:

- The Oracle Streams replication environment will be bi-directional.
- A Data Pump network import will be performed during instantiation.
- The destination database is the capture database for downstream capture of source database changes.

- The RMAN DUPLICATE or CONVERT DATABASE command will be used for database instantiation.

  This database link is required because the POST_INSTANTIATION_SETUP procedure with a non-NULL setting for the instantiation_scn parameter runs the SET_GLOBAL_INSTANTIATION_SCN procedure in the DBMS_APPLY_ADM package at the destination database. The SET_GLOBAL_INSTANTIATION_SCN procedure requires the database link. This database link must be created after the RMAN instantiation and before running the POST_INSTANTIATION_SETUP procedure.

In each of these cases, the name of the database link must match the global name of the source database.

If a third database is the capture database for downstream capture of source database changes, then the following database links are also required:

- A database link is required from the third database to the source database. The name of the database link must match the global name of the source database.
- A database link is required from the third database to the destination database. The name of the database link must match the global name of the destination database.

Each database link should be created in the Oracle Streams administrator's schema. For example, if the global name of the source database is dbs1.example.com, the global name of the destination database is dbs2.example.com, and the Oracle Streams administrator is strmadmin at each database, then the following statement creates the database link from the source database to the destination database:

```
CONNECT strmadmin@dbs1.example.com
Enter password: password

CREATE DATABASE LINK dbs2.example.com CONNECT TO strmadmin
   IDENTIFIED BY password USING 'dbs2.example.com';
```

If a database link is required from the destination database to the source database, then the following statement creates this database link:

```
CONNECT strmadmin@dbs2.example.com
Enter password: password

CREATE DATABASE LINK dbs1.example.com CONNECT TO strmadmin
   IDENTIFIED BY password USING 'dbs1.example.com';
```

If a third database is the capture database, then a database link is required from the third database to the source and destination databases. For example, if the third database is dbs3.example.com, then the following statements create the database links from the third database to the source and destination databases:

```
CONNECT strmadmin@dbs3.example.com
Enter password: password

CREATE DATABASE LINK dbs1.example.com CONNECT TO strmadmin
   IDENTIFIED BY password USING 'dbs1.example.com';

CREATE DATABASE LINK dbs2.example.com CONNECT TO strmadmin
   IDENTIFIED BY password USING 'dbs2.example.com';
```

If an RMAN database instantiation is performed, then the database link at the source database is copied to the destination database during instantiation. This copied

database link should be dropped at the destination database. In this case, if the replication is bi-directional, and a database link from the destination database to the source database is required, then this database link should be created after the instantiation.

See Also:

- *Oracle Database 2 Day DBA*
- *Oracle Database 2 Day + Data Replication and Integration Guide* for instructions about creating database links using Oracle Enterprise Manager
- *Oracle Database Administrator's Guide* for more information about database links
- *Oracle Database Heterogeneous Connectivity User's Guide* for information about communication between an Oracle database and a non-Oracle database
- "Decide Whether Changes Are Allowed at One Database or at Multiple Databases" on page 1-7
- "Decide Whether to Configure Local or Downstream Capture for the Source Database" on page 1-5

Ensuring That Each Source Database Is In ARCHIVELOG Mode

In an Oracle Streams replication environment, each source database that generates changes that will be captured by a capture process must be in `ARCHIVELOG` mode. For downstream capture processes, the downstream database also must run in `ARCHIVELOG` mode if you plan to configure a real-time downstream capture process. The downstream database does not need to run in `ARCHIVELOG` mode if you plan to run only archived-log downstream capture processes on it.

If you are configuring Oracle Streams in an Oracle Real Application Clusters (Oracle RAC) environment, then the archive log files of all threads from all instances must be available to any instance running a capture process. This requirement pertains to both local and downstream capture processes.

Note: Synchronous capture does not require `ARCHIVELOG` mode.

See Also:

- *Oracle Database Administrator's Guide* for instructions about running a database in `ARCHIVELOG` mode
- "Decide Whether Changes Are Allowed at One Database or at Multiple Databases" on page 1-7
- "Decide Whether to Configure Local or Downstream Capture for the Source Database" on page 1-5

Setting Initialization Parameters Relevant to Oracle Streams

Some initialization parameters are important for the configuration, operation, reliability, and performance of an Oracle Streams environment. Set these parameters appropriately for your Oracle Streams environment.

Table 1–2 describes the initialization parameters that are relevant to Oracle Streams. This table specifies whether each parameter is modifiable. A modifiable initialization parameter can be modified using the `ALTER SYSTEM` statement while an instance is running. Some modifiable parameters can also be modified for a single session using the `ALTER SESSION` statement.

Table 1–2 Initialization Parameters Relevant to Oracle Streams

Parameter	Values	Description
`COMPATIBLE`	**Default:** `11.2.0` **Range:** `10.0.0` to default release **Modifiable?:** No	This parameter specifies the release with which the Oracle server must maintain compatibility. Oracle servers with different compatibility levels can interoperate. To use the new Oracle Streams features introduced in Oracle Database 11*g* Release 2, this parameter must be set to `11.2.0` or higher.
`GLOBAL_NAMES`	**Default:** `false` **Range:** `true` or `false` **Modifiable?:** Yes	Specifies whether a database link is required to have the same name as the database to which it connects. To use Oracle Streams to share information between databases, set this parameter to `true` at each database that is participating in your Oracle Streams environment.
`LOG_ARCHIVE_CONFIG`	**Default:** `'SEND, RECEIVE, NODG_CONFIG'` **Range**: Values: ▪ `SEND` ▪ `NOSEND` ▪ `RECEIVE` ▪ `NORECEIVE` ▪ `DG_CONFIG` ▪ `NODG_CONFIG` **Modifiable?:** Yes	Enables or disables the sending of redo logs to remote destinations and the receipt of remote redo logs, and specifies the unique database names (`DB_UNIQUE_NAME`) for each database in the Data Guard configuration To use downstream capture and copy the redo data to the downstream database using redo transport services, specify the `DB_UNIQUE_NAME` of the source database and the downstream database using the `DG_CONFIG` attribute. This parameter must be set at both the source database and the downstream database.
`LOG_ARCHIVE_DEST_`*n*	**Default:** None **Range**: None **Modifiable?:** Yes	Defines up to 31 log archive destinations, where *n* is `1`, `2`, `3`, ... `31`. To use downstream capture and copy the redo data to the downstream database using redo transport services, at least one log archive destination must be set at the site running the downstream capture process.

Table 1–2 (Cont.) Initialization Parameters Relevant to Oracle Streams

Parameter	Values	Description
LOG_ARCHIVE_DEST_STATE_*n*	**Default:** enable **Range**: One of the following: ▪ alternate ▪ defer ▪ enable **Modifiable?:** Yes	Specifies the availability state of the corresponding destination. The parameter suffix (1 through 31) specifies one of the corresponding LOG_ARCHIVE_DEST_*n* destination parameters. To use downstream capture and copy the redo data to the downstream database using redo transport services, ensure that the destination that corresponds to the LOG_ARCHIVE_DEST_*n* destination for the downstream database is set to enable.
LOG_BUFFER	**Default:** 5 MB to 32 MB depending on configuration **Range:** Operating system-dependent **Modifiable?:** No	Specifies the amount of memory (in bytes) that Oracle uses when buffering redo entries to a redo log file. Redo log entries contain a record of the changes that have been made to the database block buffers. If an Oracle Streams capture process is running on the database, then set this parameter properly so that the capture process reads redo log records from the redo log buffer rather than from the hard disk.
MEMORY_MAX_TARGET	**Default:** 0 **Range:** 0 to the physical memory size available to Oracle Database **Modifiable?:** No	Specifies the maximum systemwide usable memory for an Oracle database. If the MEMORY_TARGET parameter is set to a nonzero value, then set this parameter to a large nonzero value if you must specify the maximum memory usage of the Oracle database. **See Also:** "Configuring the Oracle Streams Pool" on page 1-25
MEMORY_TARGET	**Default:** 0 **Range:** 152 MB to MEMORY_MAX_TARGET setting **Modifiable?:** Yes	Specifies the systemwide usable memory for an Oracle database. Oracle recommends enabling the autotuning of the memory usage of an Oracle database by setting MEMORY_TARGET to a large nonzero value (if this parameter is supported on your platform). **See Also:** "Configuring the Oracle Streams Pool" on page 1-25
OPEN_LINKS	**Default:** 4 **Range:** 0 to 255 **Modifiable?:** No	Specifies the maximum number of concurrent open connections to remote databases in one session. These connections include database links, plus external procedures and cartridges, each of which uses a separate process. In an Oracle Streams environment, ensure that this parameter is set to the default value of 4 or higher.

Table 1–2 (Cont.) Initialization Parameters Relevant to Oracle Streams

Parameter	Values	Description
PROCESSES	**Default:** 100 **Range:** 6 to operating system-dependent **Modifiable?:** No	Specifies the maximum number of operating system user processes that can simultaneously connect to Oracle. Ensure that the value of this parameter allows for all background processes, such as locks and slave processes. In Oracle Streams, capture processes, apply processes, XStream inbound servers, and XStream outbound servers use background processes. Propagations use background processes in combined capture and apply configurations. Propagations use Oracle Scheduler slave processes in configurations that do not use combined capture and apply.
SESSIONS	**Default:** Derived from: (1.5 * PROCESSES) + 22 **Range:** 1 to 2^{31} **Modifiable?:** No	Specifies the maximum number of sessions that can be created in the system. To run one or more capture processes, apply processes, XStream outbound servers, or XStream inbound servers in a database, you might need to increase the size of this parameter. Each background process in a database requires a session.
SGA_MAX_SIZE	**Default:** Initial size of SGA at startup **Range:** 0 to operating system-dependent **Modifiable?:** No	Specifies the maximum size of System Global Area (SGA) for the lifetime of a database instance. If the SGA_TARGET parameter is set to a nonzero value, then set this parameter to a large nonzero value if you must specify the SGA size. **See Also:** "Configuring the Oracle Streams Pool" on page 1-25
SGA_TARGET	**Default:** 0 (SGA autotuning is disabled) **Range:** 64 MB to operating system-dependent **Modifiable?:** Yes	Specifies the total size of all System Global Area (SGA) components. If MEMORY_MAX_TARGET and MEMORY_TARGET are set to 0 (zero), then Oracle recommends enabling the autotuning of SGA memory by setting SGA_TARGET to a large nonzero value. If this parameter is set to a nonzero value, then the size of the Oracle Streams pool is managed by Automatic Shared Memory Management. **See Also:** "Configuring the Oracle Streams Pool" on page 1-25

Table 1–2 (Cont.) Initialization Parameters Relevant to Oracle Streams

Parameter	Values	Description
SHARED_POOL_SIZE	**Default:** When SGA_TARGET is set to a nonzero value: If the parameter is not specified, then the default is 0 (internally determined by Oracle Database). If the parameter is specified, then the user-specified value indicates a minimum value for the shared memory pool. When SGA_TARGET is not set (32-bit platforms): 64 MB, rounded up to the nearest granule size. When SGA_TARGET is not set (64-bit platforms): 128 MB, rounded up to the nearest granule size. **Range:** The granule size to operating system-dependent **Modifiable?:** Yes	Specifies (in bytes) the size of the shared pool. The shared pool contains shared cursors, stored procedures, control structures, and other structures. If the MEMORY_MAX_TARGET, MEMORY_TARGET, SGA_TARGET, and STREAMS_POOL_SIZE initialization parameters are set to zero, then Oracle Streams transfers an amount equal to 10% of the shared pool from the buffer cache to the Oracle Streams pool. **See Also:**"Configuring the Oracle Streams Pool" on page 1-25

Table 1–2 (Cont.) Initialization Parameters Relevant to Oracle Streams

Parameter	Values	Description
STREAMS_POOL_SIZE	**Default:** 0 **Range:** 0 to operating system-dependent limit **Modifiable?:** Yes	Specifies (in bytes) the size of the Oracle Streams pool. The Oracle Streams pool contains buffered queue messages. In addition, the Oracle Streams pool is used for internal communications during parallel capture and apply. If the MEMORY_TARGET or MEMORY_MAX_TARGET initialization parameter is set to a nonzero value, then the Oracle Streams pool size is set by Automatic Memory Management, and STREAMS_POOL_SIZE specifies the minimum size. If the SGA_TARGET initialization parameter is set to a nonzero value, then the Oracle Streams pool size is set by Automatic Shared Memory Management, and STREAMS_POOL_SIZE specifies the minimum size. This parameter is modifiable. If this parameter is reduced to zero when an instance is running, then Oracle Streams processes and jobs might not run. Ensure that there is enough memory to accommodate the Oracle Streams components. The following are the minimum requirements: ■ 15 MB for each capture process parallelism ■ 250 MB or more for each buffered queue. The buffered queue is where the buffered messages are stored. ■ 1 MB for each apply process parallelism ■ 1 MB for each XStream outbound server ■ 1 MB for each XStream inbound server parallelism For example, if parallelism is set to 3 for a capture process, then at least 45 MB is required for the capture process. If a database has two buffered queues, then at least 20 MB is required for the buffered queues. If parallelism is set to 4 for an apply process, then at least 4 MB is required for the apply process. You can use the V$STREAMS_POOL_ADVICE dynamic performance view to determine an appropriate setting for this parameter. **See Also:** "Configuring the Oracle Streams Pool" on page 1-25

Table 1–2 (Cont.) Initialization Parameters Relevant to Oracle Streams

Parameter	Values	Description
TIMED_STATISTICS	**Default:** If STATISTICS_LEVEL is set to TYPICAL or ALL, then true If STATISTICS_LEVEL is set to BASIC, then false The default for STATISTICS_LEVEL is TYPICAL. **Range:** true or false **Modifiable?:** Yes	Specifies whether statistics related to time are collected. To collect elapsed time statistics in the dynamic performance views related to Oracle Streams, set this parameter to true. The views that include elapsed time statistics include: V$STREAMS_CAPTURE, V$STREAMS_APPLY_COORDINATOR, V$STREAMS_APPLY_READER, V$STREAMS_APPLY_SERVER.
UNDO_RETENTION	**Default:** 900 **Range:** 0 to 2^{31} - 1 **Modifiable?:** Yes	Specifies (in seconds) the amount of committed undo information to retain in the database. For a database running one or more capture processes, ensure that this parameter is set to specify an adequate undo retention period. If you run one or more capture processes and you are unsure about the proper setting, then try setting this parameter to at least 3600. If you encounter "snapshot too old" errors, then increase the setting for this parameter until these errors cease. Ensure that the undo tablespace has enough space to accommodate the UNDO_RETENTION setting.

See Also:

- *Oracle Database 2 Day DBA* for information about setting initialization parameters
- *Oracle Database Reference* for more information about these initialization parameters
- *Oracle Data Guard Concepts and Administration* for more information about the LOG_ARCHIVE_DEST_*n* parameter
- *Oracle Database Administrator's Guide* for more information about the UNDO_RETENTION parameter
- *Oracle Database XStream Guide*

Configuring the Oracle Streams Pool

The **Oracle Streams pool** is a portion of memory in the System Global Area (SGA) that is used by Oracle Streams. The Oracle Streams pool stores buffered queue messages in memory, and it provides memory for capture processes, apply processes, XStream outbound servers, and XStream inbound servers. The Oracle Streams pool always stores LCRs captured by a capture process, and it stores LCRs and messages that are enqueued into a buffered queue by applications.

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

- Messages are enqueued into a buffered queue.

  Oracle Streams components manipulate messages in a buffered queue. These components include capture processes, propagations, apply processes, XStream outbound servers, and XStream inbound servers. Also, Data Pump export and import operations initialize the Oracle Streams pool because these operations use buffered queues.

- Messages are dequeued from a persistent queue in a configuration that does not use Oracle Real Application Clusters (Oracle RAC).

  The Oracle Streams pool is used to optimize dequeue operations from persistent queues. The Oracle Streams pool is not used to optimize dequeue operations from persistent queues in an Oracle RAC configuration.

- A capture process is started.
- A propagation is created.
- An apply process is started.
- An XStream outbound server is started.
- An XStream inbound server is started.

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

- Using Automatic Memory Management to Set the Oracle Streams Pool Size
- Using Automatic Shared Memory Management to Set the Oracle Streams Pool Size
- Setting the Oracle Streams Pool Size Manually
- Using the Default Setting for the Oracle Streams Pool Size

Note: If the Oracle Streams pool cannot be initialized, then an `ORA-00832` error is returned. If this happens, then first ensure that there is enough space in the SGA for the Oracle Streams pool. If necessary, reset the `SGA_MAX_SIZE` initialization parameter to increase the SGA size. Next, set one or more of the following initialization parameters: `MEMORY_TARGET`, `MEMORY_MAX_TARGET`, `SGA_TARGET`, and `STREAMS_POOL_SIZE`.

See Also:

- "Setting Initialization Parameters Relevant to Oracle Streams" on page 1-19
- *Oracle Streams Concepts and Administration*
- *Oracle Database XStream Guide*

Using Automatic Memory Management to Set the Oracle Streams Pool Size

The Automatic Memory Management feature automatically manages the size of the Oracle Streams pool when the `MEMORY_TARGET` or `MEMORY_MAX_TARGET` initialization parameter is set to a nonzero value. When you use Automatic Memory Management, you can still set the following initialization parameters:

- If the `SGA_TARGET` initialization parameter also is set to a nonzero value, then Automatic Memory Management uses this value as a minimum for the system global area (SGA).

- If the STREAMS_POOL_SIZE initialization parameter also is set to a nonzero value, then Automatic Memory Management uses this value as a minimum for the Oracle Streams pool.

The current memory allocated to Oracle Streams pool by Automatic Memory Management can be viewed by querying the V$MEMORY_DYNAMIC_COMPONENTS view.

Note: Currently, the MEMORY_TARGET and MEMORY_MAX_TARGET initialization parameters are not supported on some platforms.

See Also:

- *Oracle Database Administrator's Guide*
- *Oracle Database Reference*

Using Automatic Shared Memory Management to Set the Oracle Streams Pool Size

The Automatic Shared Memory Management feature automatically manages the size of the Oracle Streams pool when the following conditions are met:

- The MEMORY_TARGET and MEMORY_MAX_TARGET initialization parameters are both set to 0 (zero).
- SGA_TARGET initialization parameter is set to a nonzero value.

If you are using Automatic Shared Memory Management and the STREAMS_POOL_SIZE initialization parameter also is set to a nonzero value, then Automatic Shared Memory Management uses this value as a minimum for the Oracle Streams pool. You can set a minimum size if your environment needs a minimum amount of memory in the Oracle Streams pool to function properly. The current memory allocated to Oracle Streams pool by Automatic Shared Memory Management can be viewed by querying the V$SGA_DYNAMIC_COMPONENTS view.

See Also:

- *Oracle Database Administrator's Guide*
- *Oracle Database Reference*

Setting the Oracle Streams Pool Size Manually

The Oracle Streams pool size is the value specified by the STREAMS_POOL_SIZE parameter, in bytes, if the following conditions are met.

- The MEMORY_TARGET, MEMORY_MAX_TARGET, and SGA_TARGET initialization parameters are all set to 0 (zero).
- The STREAMS_POOL_SIZE initialization parameter is set to a nonzero value.

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

See Also: *Oracle Streams Concepts and Administration*

Using the Default Setting for the Oracle Streams Pool Size

The Oracle Streams pool size is set by default if all of the following parameters are set to 0 (zero): MEMORY_TARGET, MEMORY_MAX_TARGET, SGA_TARGET, and STREAMS_

POOL_SIZE. When the Oracle Streams pool size is set by default, the first use of Oracle Streams in a database transfers an amount of memory equal to 10% of the shared pool from the buffer cache to the Oracle Streams pool. The buffer cache is set by the DB_CACHE_SIZE initialization parameter, and the shared pool size is set by the SHARED_POOL_SIZE initialization parameter.

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

- DB_CACHE_SIZE is set to 100 MB.
- SHARED_POOL_SIZE is set to 80 MB.
- MEMORY_TARGET, MEMORY_MAX_TARGET, SGA_TARGET, and STREAMS_POOL_SIZE are all set to zero.

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

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

> **See Also:** "Setting Initialization Parameters Relevant to Oracle Streams" on page 1-19 for more information about the STREAMS_POOL_SIZE initialization parameter

Specifying Supplemental Logging

When you use a capture process to capture changes, 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. Supplemental logging places additional information in the redo log for these columns. A capture process captures this additional information and places it in logical change records (LCRs), and an apply process might need this additional information to apply changes properly.

This section contains these topics:

- Required Supplemental Logging in an Oracle Streams Replication Environment
- Specifying Table Supplemental Logging Using Unconditional Log Groups
- Specifying Table Supplemental Logging Using Conditional Log Groups
- Dropping a Supplemental Log Group
- Specifying Database Supplemental Logging of Key Columns
- Dropping Database Supplemental Logging of Key Columns
- Procedures That Automatically Specify Supplemental Logging

> **Note:** Supplemental logging is not required when synchronous capture is used to capture changes to database objects.

> **See Also:** *Oracle Streams Concepts and Administration* for queries that show supplemental logging specifications

Required Supplemental Logging in an Oracle Streams Replication Environment

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 of a particular table. If you use table supplemental logging, then you can choose between two types of log groups: unconditional log groups and conditional log groups.

Unconditional log groups log the before images of specified columns when the table is changed, regardless of whether the change affected any of the specified columns. Unconditional log groups are sometimes referred to as "always log groups." **Conditional log groups** log the before images of all specified columns only if at least one of the columns in the log group is changed.

Supplementing logging at the database level, unconditional log groups at the table level, and conditional log groups at the table level determine which old values are logged for a change.

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 column at a destination database that comes from multiple columns at the source database must be conditionally logged. Supplemental logging does not need to be specified if a unique constraint column 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 column at a destination database that comes from multiple columns at the source database must be conditionally logged. Supplemental logging does not need to be specified if the foreign key column 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 bitmap index column at a destination database that comes from multiple columns at the source database must be conditionally logged. Supplemental logging does not need to be specified if the bitmap index column comes from a single column at the source database.
- Any columns at the source database that are used as 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 multiple columns at the source database are used in the column list at the destination database.
- Any columns at the source database that are used by a statement DML handler, change handler, procedure DML handler, or error handler 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.

- Any columns at the source database that are specified in a value dependency virtual dependency definition at a destination database 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: Columns of the following data types cannot be part of a supplemental log group: LOB, `LONG`, `LONG RAW`, user-defined types (including object types, `REF`s, varrays, nested tables), and Oracle-supplied types (including `Any` types, XML types, spatial types, and media types).

See Also:

- "Column Lists" on page 9-9
- "Decide Whether the Replication Environment Will Have Nonidentical Replicas" on page 1-8 for information about rule-based transformations
- "Decide Whether the Replication Environment Will Use Apply Handlers" on page 1-9
- *Oracle Database SQL Language Reference* for more information about data types

Specifying Table Supplemental Logging Using Unconditional Log Groups

The following sections describe creating an unconditional supplemental log group:

- Specifying an Unconditional Supplemental Log Group for Primary Key Column(s)
- Specifying an Unconditional Supplemental Log Group for All Table Columns
- Specifying an Unconditional Supplemental Log Group that Includes Selected Columns

Specifying an Unconditional Supplemental Log Group for Primary Key Column(s) To specify an unconditional supplemental log group that only includes the primary key column(s) for a table, use an `ALTER TABLE` statement with the `PRIMARY KEY` option in the `ADD SUPPLEMENTAL LOG DATA` clause.

For example, the following statement adds the primary key column of the `hr.regions` table to an unconditional log group:

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

The log group has a system-generated name.

Specifying an Unconditional Supplemental Log Group for All Table Columns To specify an unconditional supplemental log group that includes all of the columns in a table, use

an ALTER TABLE statement with the ALL option in the ADD SUPPLEMENTAL LOG DATA clause.

For example, the following statement adds all of the columns in the hr.regions table to an unconditional log group:

```
ALTER TABLE hr.regions ADD SUPPLEMENTAL LOG DATA (ALL) COLUMNS;
```

The log group has a system-generated name.

Specifying an Unconditional Supplemental Log Group that Includes Selected Columns To specify an unconditional supplemental log group that contains columns that you select, use an ALTER TABLE statement with the ALWAYS specification for the ADD SUPPLEMENTAL LOG GROUP clause.These log groups can include key columns, if necessary.

For example, the following statement adds the department_id column and the manager_id 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, manager_id) ALWAYS;
```

The ALWAYS specification makes this log group an unconditional log group.

Specifying Table Supplemental Logging Using Conditional Log Groups

The following sections describe creating a conditional log group:

- Specifying a Conditional Log Group Using the ADD SUPPLEMENTAL LOG DATA Clause
- Specifying a Conditional Log Group Using the ADD SUPPLEMENTAL LOG GROUP Clause

Specifying a Conditional Log Group Using the ADD SUPPLEMENTAL LOG DATA Clause You can use the following options in the ADD SUPPLEMENTAL LOG DATA clause of an ALTER TABLE statement:

- The FOREIGN KEY option creates a conditional log group that includes the foreign key column(s) in the table.
- The UNIQUE option creates a conditional log group that includes the unique key column(s) and bitmap index column(s) in the table.

If you specify multiple options in a single ALTER TABLE statement, then a separate conditional log group is created for each option.

For example, the following statement creates two conditional log groups:

```
ALTER TABLE hr.employees ADD SUPPLEMENTAL LOG DATA
  (UNIQUE, FOREIGN KEY) COLUMNS;
```

One conditional log group includes the unique key columns and bitmap index columns for the table, and the other conditional log group includes the foreign key columns for the table. Both log groups have a system-generated name.

> **Note:** Specifying the UNIQUE option does not enable supplemental logging of bitmap join index columns.

Specifying a Conditional Log Group Using the ADD SUPPLEMENTAL LOG GROUP Clause To specify a conditional supplemental log group that includes any columns you choose to add, you can use 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 conditional log 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, unique key, bitmap index, and foreign key columns in a source database. You might choose this option if you configure a capture process to capture changes to an entire database. To specify supplemental logging for all primary key, unique key, bitmap index, and foreign key columns in a source database, issue the following SQL statement:

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA
   (PRIMARY KEY, UNIQUE, FOREIGN KEY) COLUMNS;
```

If your primary key, unique key, bitmap index, and foreign 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 key, unique key, bitmap index, and foreign key columns at all destination databases. When you specify the `PRIMARY KEY` option, all columns of a row's primary key are placed in the redo log file any time the table is modified (unconditional logging). When you specify the `UNIQUE` option, any columns in a row's unique key and bitmap index are placed in the redo log file if any column belonging to the unique key or bitmap index is modified (conditional logging). When you specify the `FOREIGN KEY` option, all columns of a row's foreign key are placed in the redo log file if any column belonging to the foreign key is modified (conditional logging).

You can omit one or more of these options. For example, if you do not want to supplementally log all of the foreign key columns in the database, then you can omit the `FOREIGN KEY` option, as in the following example:

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

In additional to `PRIMARY KEY`, `UNIQUE`, and `FOREIGN KEY`, you can also use the `ALL` option. The `ALL` option specifies that, when a row is changed, all the columns of that row (except for LOB, `LONG`, `LONG RAW`, user-defined type, and Oracle-supplied type columns) are placed in the redo log file (unconditional logging).

Supplemental logging statements are cumulative. If you issue two consecutive ALTER DATABASE ADD SUPPLEMENTAL LOG DATA commands, each with a different identification key, then both keys are supplementally logged.

> **Note:** Specifying the UNIQUE option does not enable supplemental logging of bitmap join index columns.

> **See Also:** *Oracle Database SQL Language Reference* for information about data types

Dropping Database Supplemental Logging of Key Columns

To drop supplemental logging for all primary key, unique key, bitmap index, and foreign key columns in a source database, issue the ALTER DATABASE DROP SUPPLEMENTAL LOG DATA statement. To drop database supplemental logging for all primary key, unique key, bitmap index, and foreign key columns, issue the following SQL statement:

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

> **Note:** Dropping database supplemental logging of key columns does not affect any existing table-level supplemental log groups.

Procedures That Automatically Specify Supplemental Logging

The following procedures in the DBMS_CAPTURE_ADM package automatically specify supplemental logging:

- BUILD
- PREPARE_GLOBAL_INSTANTIATION
- PREPARE_SCHEMA_INSTANTIATION
- PREPARE_TABLE_INSTANTIATION

The BUILD procedure automatically specifies database supplemental logging by running the ALTER DATABASE ADD SUPPLEMENTAL LOG DATA statement. In most cases, the BUILD procedure is run automatically when a capture process is created.

The PREPARE_GLOBAL_INSTANTIATION, PREPARE_SCHEMA_INSTANTIATION, and PREPARE_TABLE_INSTANTIATION procedures automatically specify supplemental logging of the primary key, unique key, bitmap index, and foreign key columns in the tables prepared for instantiation.

Certain procedures in the DBMS_STREAMS_ADM package automatically run a procedure listed previously. See "DBMS_STREAMS_ADM Package Procedures Automatically Prepare Objects" on page 8-4 for information.

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

Configuring Log File Transfer to a Downstream Capture Database

If you decided to use a local capture process at the source database, then log file transfer is not required. However, if you decided to use downstream capture that uses

redo transport services to transfer archived redo log files to the downstream database automatically, then configure log file transfer from the source database to the capture database before configuring the replication environment. See "Decide Whether to Configure Local or Downstream Capture for the Source Database" on page 1-5 for information about the decision.

You must complete the steps in this section if you plan to configure downstream capture using either of the following methods:

- Running a configuration procedure in the `DBMS_STREAMS_ADM` supplied PL/SQL package to configure replication between two databases
- Configuring each Oracle Streams component separately

See "Decide How to Configure the Replication Environment" on page 1-10 for information about these methods.

> **Tip:** You can use Oracle Enterprise Manager to configure log file transfer and a downstream capture process. See *Oracle Database 2 Day + Data Replication and Integration Guide* for instructions.

Complete the following steps to prepare the source database to transfer its redo log files to the capture database, and to prepare the capture database to accept these redo log files:

1. Configure Oracle Net so that the source database can communicate with the downstream database.

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

2. Configure authentication at both databases to support the transfer of redo data.

   Redo transport sessions are authenticated using either the Secure Sockets Layer (SSL) protocol or a remote login password file. If the source database has a remote login password file, then copy it to the appropriate directory on the downstream capture database system. The password file must be the same at the source database and the downstream capture database.

   > **See Also:** *Oracle Data Guard Concepts and Administration* for detailed information about authentication requirements for redo transport

3. At the source database, set the following initialization parameters to configure redo transport services to transmit redo data from the source database to the downstream database:

   - `LOG_ARCHIVE_DEST_n` - Configure at least one `LOG_ARCHIVE_DEST_n` initialization parameter to transmit redo data to the downstream database. Set the following attributes of this parameter in the following way:
     - `SERVICE` - Specify the network service name of the downstream database.
     - `ASYNC` or `SYNC` - Specify a redo transport mode.

       The advantage of specifying `ASYNC` is that it results in little or no effect on the performance of the source database. If the source database is running Oracle Database 10*g* Release 1 or later, then `ASYNC` is recommended to avoid affecting source database performance if the downstream database or network is performing poorly.

       The advantage of specifying `SYNC` is that redo data is sent to the downstream database faster then when `ASYNC` is specified. Also, specifying

SYNC AFFIRM results in behavior that is similar to MAXIMUM AVAILABILITY standby protection mode. Note that specifying an ALTER DATABASE STANDBY DATABASE TO MAXIMIZE AVAILABILITY SQL statement has no effect on an Oracle Streams capture process.

– NOREGISTER - Specify this attribute so that the location of the archived redo log files is not recorded in the downstream database control file.

– VALID_FOR - Specify either (ONLINE_LOGFILE,PRIMARY_ROLE) or (ONLINE_LOGFILE,ALL_ROLES).

– TEMPLATE - If you are configuring an archived-log downstream capture process, then specify a directory and format template for archived redo logs at the downstream database. The TEMPLATE attribute overrides the LOG_ARCHIVE_FORMAT initialization parameter settings at the downstream database. The TEMPLATE attribute is valid only with remote destinations. Ensure that the format uses all of the following variables at each source database: %t, %s, and %r.

  Do not specify the TEMPLATE attribute if you are configuring a real-time downstream capture process.

– DB_UNIQUE_NAME - The unique name of the downstream database. Use the name specified for the DB_UNIQUE_NAME initialization parameter at the downstream database.

The following example is a LOG_ARCHIVE_DEST_*n* setting that specifies the downstream database dbs2 for a real-time downstream capture process:

```
LOG_ARCHIVE_DEST_2='SERVICE=DBS2.EXAMPLE.COM ASYNC NOREGISTER
   VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)
   DB_UNIQUE_NAME=dbs2'
```

The following example is a LOG_ARCHIVE_DEST_*n* setting that specifies the downstream database dbs2 for an archived-log downstream capture process:

```
LOG_ARCHIVE_DEST_2='SERVICE=DBS2.EXAMPLE.COM ASYNC NOREGISTER
   VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)
   TEMPLATE=/usr/oracle/log_for_dbs1/dbs1_arch_%t_%s_%r.log
   DB_UNIQUE_NAME=dbs2'
```

See "Decide Whether to Configure Local or Downstream Capture for the Source Database" on page 1-5 for information about the differences between real-time and archived-log downstream capture.

Tip: If you are configuring an archived-log downstream capture process, then specify a value for the TEMPLATE attribute that keeps log files from a remote source database separate from local database log files. In addition, if the downstream database contains log files from multiple source databases, then the log files from each source database should be kept separate from each other.

- LOG_ARCHIVE_DEST_STATE_*n* - Set this initialization parameter that corresponds with the LOG_ARCHIVE_DEST_*n* parameter for the downstream database to ENABLE.

  For example, if the LOG_ARCHIVE_DEST_2 initialization parameter is set for the downstream database, then set the LOG_ARCHIVE_DEST_STATE_2 parameter in the following way:

```
LOG_ARCHIVE_DEST_STATE_2=ENABLE
```

- LOG_ARCHIVE_CONFIG - Set the DB_CONFIG attribute in this initialization parameter to include the DB_UNIQUE_NAME of the source database and the downstream database.

  For example, if the DB_UNIQUE_NAME of the source database is dbs1, and the DB_UNIQUE_NAME of the downstream database is dbs2, then specify the following parameter:

  ```
  LOG_ARCHIVE_CONFIG='DG_CONFIG=(dbs1,dbs2)'
  ```

  By default, the LOG_ARCHIVE_CONFIG parameter enables a database to both send and receive redo.

  See Also: *Oracle Database Reference* and *Oracle Data Guard Concepts and Administration* for more information about these initialization parameters

4. At the downstream database, set the DB_CONFIG attribute in the LOG_ARCHIVE_CONFIG initialization parameter to include the DB_UNIQUE_NAME of the source database and the downstream database.

   For example, if the DB_UNIQUE_NAME of the source database is dbs1, and the DB_UNIQUE_NAME of the downstream database is dbs2, then specify the following parameter:

   ```
   LOG_ARCHIVE_CONFIG='DG_CONFIG=(dbs1,dbs2)'
   ```

   By default, the LOG_ARCHIVE_CONFIG parameter enables a database to both send and receive redo.

5. If you reset any initialization parameters while the instance was running at a database in Step 3 or Step 4, then you might want to reset them in the initialization parameter file as well, so that the new values are retained when the database is restarted.

   If you did not reset the initialization parameters while the instance was running, but instead reset them in the initialization parameter file in Step 3 or Step 4, then restart the database. The source database must be open when it sends redo log files to the downstream database, because the global name of the source database is sent to the downstream database only if the source database is open.

When these steps are complete, you are ready to perform one of the following tasks:

- Configure an archived-log downstream capture process. In this case, see the instructions in the following sections:
  - "Configuring Replication Using the DBMS_STREAMS_ADM Package" on page 2-3
  - "Configuring an Archived-Log Downstream Capture Process" on page 5-10
- Add standby redo logs files at the downstream database for a real-time downstream capture process. In this case, see the instructions in "Adding Standby Redo Logs for Real-Time Downstream Capture" on page 1-36.

Adding Standby Redo Logs for Real-Time Downstream Capture

The example in this section adds standby redo logs at a downstream database. Standby redo logs are required to configure a real-time downstream capture process.

In the example, the source database is `dbs1.example.com` and the downstream database is `dbs2.example.com`

See "Decide Whether to Configure Local or Downstream Capture for the Source Database" on page 1-5 for information about the differences between real-time and archived-log downstream capture. The steps in this section are required only if you are configuring real-time downstream capture. If you are configuring archived-log downstream capture, then do not complete the steps in this section.

> **Tip:** You can use Oracle Enterprise Manager to configure real-time downstream capture. See *Oracle Database 2 Day + Data Replication and Integration Guide* for instructions.

Complete the following steps to add a standby redo log at the downstream database:

1. Complete the steps in "Configuring Log File Transfer to a Downstream Capture Database" on page 1-33.
2. At the downstream database, set the following initialization parameters to configure archiving of the redo data generated locally:
   - Set at least one archive log destination in the `LOG_ARCHIVE_DEST_n` initialization parameter either to a directory or to the fast recovery area on the computer system running the downstream database. Set the following attributes of this parameter in the following way:
     - `LOCATION` - Specify either a valid path name for a disk directory or, to use a fast recovery area, specify `USE_DB_RECOVERY_FILE_DEST`. This location is the local destination for archived redo log files written from the standby redo logs. Log files from a remote source database should be kept separate from local database log files. See *Oracle Database Backup and Recovery User's Guide* for information about configuring a fast recovery area.
     - `VALID_FOR` - Specify either `(ONLINE_LOGFILE,PRIMARY_ROLE)` or `(ONLINE_LOGFILE,ALL_ROLES)`.

     The following example is a `LOG_ARCHIVE_DEST_n` setting for the locally generated redo data at the real-time downstream capture database:

     ```
     LOG_ARCHIVE_DEST_1='LOCATION=/home/arc_dest/local_rl_dbs2
        VALID_FOR=(ONLINE_LOGFILE,PRIMARY_ROLE)'
     ```

     A real-time downstream capture configuration should keep archived standby redo log files separate from archived online redo log files generated by the downstream database. Specify `ONLINE_LOGFILE` instead of `ALL_LOGFILES` for the redo log type in the `VALID_FOR` attribute to accomplish this.

     You can specify other attributes in the `LOG_ARCHIVE_DEST_n` initialization parameter if necessary.
   - Set the `LOG_ARCHIVE_DEST_STATE_n` initialization parameter that corresponds with the `LOG_ARCHIVE_DEST_n` parameter previously set in this step to `ENABLE`.

     For example, if the `LOG_ARCHIVE_DEST_1` initialization parameter is set, then set the `LOG_ARCHIVE_DEST_STATE_1` parameter in the following way:

     ```
     LOG_ARCHIVE_DEST_STATE_1=ENABLE
     ```

3. At the downstream database, set the following initialization parameters to configure the downstream database to receive redo data from the source database and write the redo data to the standby redo log at the downstream database:

   - Set at least one archive log destination in the `LOG_ARCHIVE_DEST_n` initialization parameter either to a directory or to the fast recovery area on the computer system running the downstream database. Set the following attributes of this parameter in the following way:

     – `LOCATION` - Specify either a valid path name for a disk directory or, to use a fast recovery area, specify `USE_DB_RECOVERY_FILE_DEST`. This location is the local destination for archived redo log files written from the standby redo logs. Log files from a remote source database should be kept separate from local database log files. See *Oracle Database Backup and Recovery User's Guide* for information about configuring a fast recovery area.

     – `VALID_FOR` - Specify either `(STANDBY_LOGFILE,PRIMARY_ROLE)` or `(STANDBY_LOGFILE,ALL_ROLES)`.

     The following example is a `LOG_ARCHIVE_DEST_n` setting for the redo data received from the source database at the real-time downstream capture database:

     ```
     LOG_ARCHIVE_DEST_2='LOCATION=/home/arc_dest/srl_dbs1
        VALID_FOR=(STANDBY_LOGFILE,PRIMARY_ROLE)'
     ```

     You can specify other attributes in the `LOG_ARCHIVE_DEST_n` initialization parameter if necessary.

   - Set the `LOG_ARCHIVE_DEST_STATE_n` initialization parameter that corresponds with the `LOG_ARCHIVE_DEST_n` parameter previously set in this step to `ENABLE`.

     For example, if the `LOG_ARCHIVE_DEST_2` initialization parameter is set for the downstream database, then set the `LOG_ARCHIVE_DEST_STATE_2` parameter in the following way:

     ```
     LOG_ARCHIVE_DEST_STATE_2=ENABLE
     ```

     See Also: *Oracle Database Reference* and *Oracle Data Guard Concepts and Administration* for more information about these initialization parameters

4. If you reset any initialization parameters while an instance was running at a database in Step 2 or 3, then you might want to reset them in the relevant initialization parameter file as well, so that the new values are retained when the database is restarted.

   If you did not reset the initialization parameters while an instance was running, but instead reset them in the initialization parameter file in Step 2 or 3, then restart the database. The source database must be open when it sends redo data to the downstream database, because the global name of the source database is sent to the downstream database only if the source database is open.

5. Create the standby redo log files.

Note: The following steps outline the general procedure for adding standby redo log files to the downstream database. The specific steps and SQL statements used to add standby redo log files depend on your environment. For example, in an Oracle Real Application Clusters (Oracle RAC) environment, the steps are different. See *Oracle Data Guard Concepts and Administration* for detailed instructions about adding standby redo log files to a database.

---

a. In SQL*Plus, connect to the source database `dbs1.example.com` as an administrative user.

See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

b. Determine the log file size used on the source database. The standby log file size must exactly match (or be larger than) the source database log file size. For example, if the source database log file size is 500 MB, then the standby log file size must be 500 MB or larger. You can determine the size of the redo log files at the source database (in bytes) by querying the `V$LOG` view at the source database.

For example, query the `V$LOG` view:

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

c. Determine the number of standby log file groups required on the downstream database. The number of standby log file groups must be at least one more than the number of online log file groups on the source database. For example, if the source database has two online log file groups, then the downstream database must have at least three standby log file groups. You can determine the number of source database online log file groups by querying the `V$LOG` view at the source database.

For example, query the `V$LOG` view:

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

d. In SQL*Plus, connect to the downstream database `dbs2.example.com` as an administrative user.

e. Use the SQL statement `ALTER DATABASE ADD STANDBY LOGFILE` to add the standby log file groups to the downstream database.

For example, assume that the source database has two online redo log file groups and is using a log file size of 500 MB. In this case, use the following statements to create the appropriate standby log file groups:

```
ALTER DATABASE ADD STANDBY LOGFILE GROUP 3
   ('/oracle/dbs/slog3a.rdo', '/oracle/dbs/slog3b.rdo') SIZE 500M;

ALTER DATABASE ADD STANDBY LOGFILE GROUP 4
   ('/oracle/dbs/slog4.rdo', '/oracle/dbs/slog4b.rdo') SIZE 500M;

ALTER DATABASE ADD STANDBY LOGFILE GROUP 5
   ('/oracle/dbs/slog5.rdo', '/oracle/dbs/slog5b.rdo') SIZE 500M;
```

f. Ensure that the standby log file groups were added successfully by running the following query:

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

You output should be similar to the following:

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

g. Ensure that log files from the source database are appearing in the location specified in the `LOCATION` attribute in Step 3. You might need to switch the log file at the source database to see files in the directory.

When these steps are complete, you are ready to configure a real-time downstream capture process. See the instructions in the following sections:

- "Configuring Replication Using the DBMS_STREAMS_ADM Package" on page 2-3
- "Configuring a Real-Time Downstream Capture Process" on page 5-7

2

Simple Oracle Streams Replication Configuration

This chapter describes simple methods for configuring Oracle Streams replication.

This chapter contains these topics:

- Configuring Replication Using the Setup Streams Replication Wizard
- Configuring Replication Using the DBMS_STREAMS_ADM Package

Configuring Replication Using the Setup Streams Replication Wizard

The Oracle Streams tool in Oracle Enterprise Manager includes a Setup Streams Replication Wizard that configures an Oracle Streams replication environment. Using this wizard, you can configure an Oracle Streams replication environment with any of the following characteristics:

- Replicate changes to the entire source database, selected schemas in the source database, selected tablespaces in the source database, selected tables in the source database, or subsets of tables in the source database
- Maintain data manipulation language (DML) changes, data definition language (DDL) changes, or both.
- One-way or bi-directional replication
- Local capture or downstream capture

The wizard walks you through the process of configuring your replication environment, and you can run the wizard multiple times to configure a replication environment with more than two databases. There are some limits to the types of replication environments that can be configured with the wizard. For example, the wizard currently cannot configure synchronous capture.

You can choose to configure the Oracle Streams replication environment immediately, or you can use the wizard to generate a script. When you generate a script, you can review the script, and edit the script if necessary, before running the script to configure the replication environment.

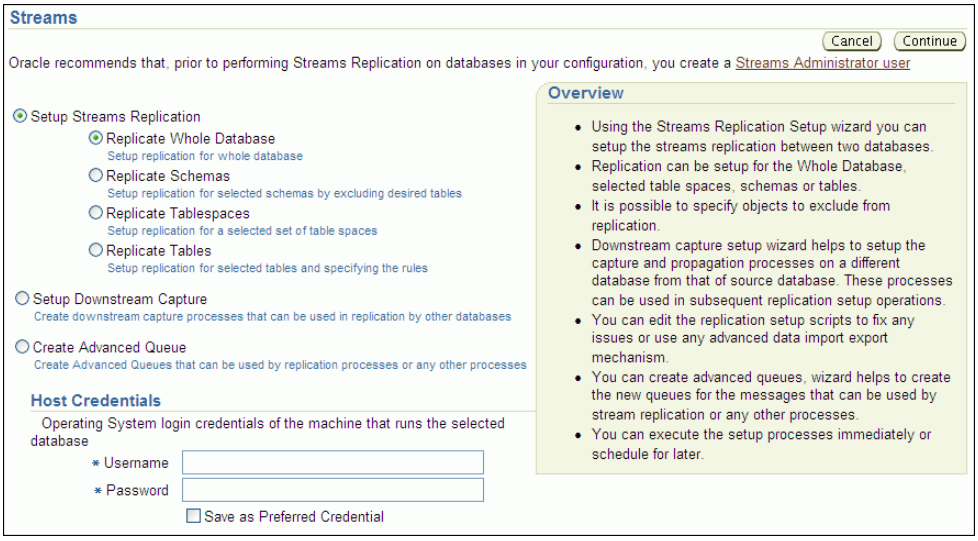
To open the wizard, complete the following steps in Enterprise Manager:

1. Review the decisions described in "Decisions to Make Before Configuring Oracle Streams Replication" on page 1-4. Make these decisions about the Oracle Streams replication environment before proceeding.
2. Complete the tasks to prepare for the Oracle Streams replication environment. See "Tasks to Complete Before Configuring Oracle Streams Replication" on page 1-13.

The wizard completes some tasks automatically, but you must complete the following tasks manually:

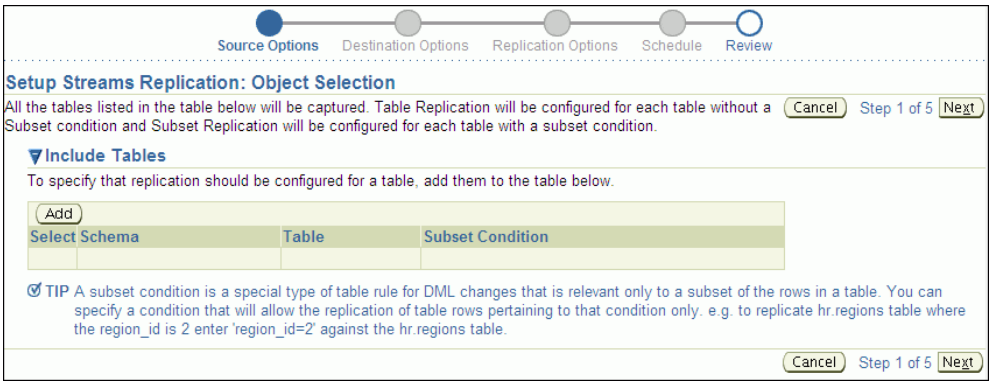
- Configure an Oracle Streams administrator at both databases. See "Configuring an Oracle Streams Administrator on All Databases" on page 1-13. If you have not done so already, you must click the **Streams Administrator user** link after you open the Streams page in Step 6 to configure the Oracle Streams administrator to manage Oracle Streams using Oracle Enterprise Manager.
- Configure network connectivity between the two databases. See "Configuring Network Connectivity and Database Links" on page 1-17.
- Ensure that any database that will be a source database is in `ARCHIVELOG` mode. If you are configuring bi-directional replication, then both databases must be in `ARCHIVELOG` mode. See "Ensuring That Each Source Database Is In ARCHIVELOG Mode" on page 1-19.
- Ensure that the initialization parameters are set properly at both databases. See "Setting Initialization Parameters Relevant to Oracle Streams" on page 1-19.
- Configure the Oracle Streams pool properly at both databases. See "Configuring the Oracle Streams Pool" on page 1-25.

3. In Oracle Enterprise Manager, log in to the database as the Oracle Streams administrator. Log in to a database that will be a source database in the replication environment.
4. Go to the Database Home page.
5. Click **Data Movement** to open the Data Movement subpage.
6. Click **Setup** in the **Streams** section to open the Streams page.



7. In the Setup Streams Replication section, select the option for the type of replication environment you want to configure.
8. In the Host Credentials section, enter the username and password for the host computer that runs the source database.
9. Click Continue to open the Setup Streams Replication wizard.

Here is the first wizard page when you select Replicate Tables on the Streams page.



Note: By default, the Setup Streams Replication Wizard configures one-way replication. To configure bi-directional replication, open the Advanced Options section on the Replication Options page and select **Setup Bi-directional replication**. If bi-directional replication is configured, then conflict resolution might be required.

See Also:

- *Oracle Database 2 Day + Data Replication and Integration Guide* for examples that configure an Oracle Streams replication environment with the wizard
- "Decide Which Type of Replication Environment to Configure" on page 1-4
- Chapter 9, "Oracle Streams Conflict Resolution"

Configuring Replication Using the DBMS_STREAMS_ADM Package

You can configure an Oracle Streams replication environment using procedures in the `DBMS_STREAMS_ADM` package.

The following sections contain information about these procedures, instructions for preparing to run one of these procedures, and examples that illustrate common scenarios:

- The Oracle Streams Replication Configuration Procedures
- Important Considerations for the Configuration Procedures
- Creating the Required Directory Objects
- Examples That Configure Two-Database Replication with Local Capture
- Examples That Configure Two-Database Replication with Downstream Capture
- Example That Configures Hub-and-Spoke Replication
- Monitoring Oracle Streams Configuration Progress

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for more information about these procedures
- *Oracle Database 2 Day + Data Replication and Integration Guide* for several examples that use these procedures to configure an Oracle Streams replication environment

The Oracle Streams Replication Configuration Procedures

The easiest way to configure an Oracle Streams replication environment is by running one of the following configuration procedures in the `DBMS_STREAMS_ADM` package:

- `MAINTAIN_GLOBAL` configures an Oracle Streams environment that replicates changes at the database level between two databases.
- `MAINTAIN_SCHEMAS` configures an Oracle Streams environment that replicates changes to specified schemas between two databases.
- `MAINTAIN_SIMPLE_TTS` clones a simple tablespace from a source database at a destination database and uses Oracle Streams to maintain this tablespace at both databases.
- `MAINTAIN_TABLES` configures an Oracle Streams environment that replicates changes to specified tables between two databases.
- `MAINTAIN_TTS` clones a set of tablespaces from a source database at a destination database and uses Oracle Streams to maintain these tablespaces at both databases.
- `PRE_INSTANTIATION_SETUP` and `POST_INSTANTIATION_SETUP` configure an Oracle Streams environment that replicates changes either at the database level or to specified tablespaces between two databases. These procedures must be used together, and instantiation actions must be performed manually, to complete the Oracle Streams replication configuration. Typically, these procedures are used to perform database maintenance operations with little or no down time. See *Oracle Streams Concepts and Administration* for more information.

These procedures configure two databases at a time, and they require you to specify one database as the source database and the other database as the destination database. They can configure a replication environment with more than two databases by running them multiple times.

Table 2–1 describes the required parameters for these procedures.

Table 2–1 Required Parameters for the Oracle Streams Replication Configuration Procedures

Parameter	Procedure	Description
`source_directory_object`	All procedures	The directory object for the directory on the computer system running the source database into which the generated Data Pump export dump file is placed. **Note:** The specified directory object cannot point to an Automatic Storage Management (ASM) disk group.
`destination_directory_object`	All procedures	The directory object for the directory on the computer system running the destination database into which the generated Data Pump export dump file is transferred. The dump file is used to instantiate the replicated database objects at the destination database. **Note:** The specified directory object cannot point to an Automatic Storage Management (ASM) disk group.
`source_database`	All procedures	The global name of the source database. The specified database must contain the database objects to be replicated.
`destination_database`	All procedures	The global name of the destination database. The database objects to be replicated are optional at the destination database. If they do not exist at the destination database, then they are instantiated by Data Pump export/import. If the local database is not the destination database, then a database link from the local database to the destination database, with the same name as the global name of the destination database, must exist and must be accessible to the user who runs the procedure.
`schema_names`	`MAINTAIN_SCHEMAS` only	The schemas to be configured for replication.
`tablespace_name`	`MAINTAIN_SIMPLE_TTS` only	The tablespace to be configured for replication.
`table_names`	`MAINTAIN_TABLES` only	The tables to be configured for replication.
`tablespace_names`	`MAINTAIN_TTS` only	The tablespaces to be configured for replication.

In addition, each of these procedures has several optional parameters. The `bi_directional` parameter is an important optional parameter. If you want changes to the replicated database objects to be captured at each database and sent to the other database, then the `bi_directional` parameter must be set to `TRUE`. The default setting for this parameter is `FALSE`. When the `bi_directional` parameter is set to `FALSE`, the procedures configure a one-way replication environment, where the changes made at the destination database are not captured.

These procedures perform several tasks to configure an Oracle Streams replication environment. These tasks include:

- Configure supplemental logging for the replicated database objects at the source database.
- If the `bi_directional` parameter is set to `TRUE`, then configure supplemental logging for the replicated database objects at the destination database.
- Instantiate the database objects at the destination database. If the database objects do not exist at the destination database, then the procedures use Data Pump export/import to instantiate them at the destination database.
- Configure a capture process to capture changes to the replicated database objects at the source database. This capture process can be a local capture process or a downstream capture process. If the procedure is run at the database specified in the `source_database` parameter, then the procedure configures a local capture process on this database. If the procedure is run at a database other than the database specified in the `source_database` parameter, then the procedure configures a downstream capture process on the database that runs the procedure.
- If the `bi_directional` parameter is set to `TRUE`, then configure a capture process to capture changes to the replicated database objects at the destination database. This capture process must be a local capture process.
- Configure one or more queues at each database to store captured changes.
- Configure a propagation to send changes made to the database objects at the source database to the destination database.
- If the `bi_directional` parameter is set to `TRUE`, then configure a propagation to send changes made to the database objects at the destination database to the source database
- Configure an apply process at the destination database to apply changes from the source database.
- If the `bi_directional` parameter is set to `TRUE`, then configure an apply process at the source database to apply changes from the destination database.
- Configure rule sets and rules for each capture process, propagation, and apply process. The rules instruct the Oracle Streams clients to replicate changes to the specified database objects.
- Set the instantiation SCN for the replicated database objects at the destination database.
- If the `bi_directional` parameter is set to `TRUE`, then set the instantiation SCN for the replicated database objects at the source database.

> **Tip:** To view all of the actions performed by one of these procedures in detail, use the procedure to generate a script, and view the script in a text editor. You can use the `perform_actions`, `script_name`, and `script_directory_object` parameters to generate a script.

These procedures always configure tags for a hub-and-spoke replication environment. The following are important considerations about these procedures and tags:

- If you are configuring a two-database replication environment, then you can use these procedures to configure it. These procedures configure tags in a two-database environment to avoid change cycling. If you plan to expand the replication environment beyond two databases in the future, then it is important to understand how the tags are configured. If the expanded database environment is not a hub-and-spoke environment, then you might need to modify the tags to avoid change cycling.

- If you are configuring a replication environment that involves three or more databases, then these procedures can only be used to configure a hub-and-spoke replication environment. These procedures configure tags in a hub-and-spoke environment to avoid change cycling.
- If you are configuring an n-way replication environment, then do not use these procedures to configure it. Change cycling might result if you do so.

Note: Currently, these configuration procedures configure only capture processes to capture changes. You cannot use these procedures to configure a replication environment that uses synchronous captures. You can configure a synchronous capture using the `ADD_TABLE_RULES` and `ADD_SUBSET_RULES` procedures in the `DBMS_STREAMS_ADM` package.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for more information about the procedures in the `DBMS_STREAMS_ADM` package
- Chapter 10, "Oracle Streams Tags"
- "Decide Which Type of Replication Environment to Configure" on page 1-4
- *Oracle Streams Concepts and Administration*

Important Considerations for the Configuration Procedures

This section describes important considerations for the configuration procedures. It also discusses several procedure parameters related to these considerations.

This section contains these topics:

- Local or Downstream Capture for the Source Database
- Perform Configuration Actions Directly or With a Script
- Oracle Streams Components Configured by These Procedures
- One-Way or Bi-Directional Replication
- Data Definition Language (DDL) Changes
- Instantiation

See Also: *Oracle Database PL/SQL Packages and Types Reference* for information about all of the parameters in these procedures

Local or Downstream Capture for the Source Database

These procedures can either configure local capture or downstream capture for the database specified in the `source_database` parameter. The database that captures changes made to the source database is called the capture database. See "Decide Whether to Configure Local or Downstream Capture for the Source Database" on page 1-5 for more information.

The database on which the procedure is run is configured as the capture database for changes made to the source database. Therefore, to configure local capture at the source database, run the procedure at the source database. To configure downstream

capture at the destination database, run the procedure at the database specified in the `destination_database` parameter. To configure downstream capture at the at a third database, run the procedure at a database that is not specified in the `source_database` or `destination_database` parameter.

If the source database or a third database is the capture database, then these procedures configure a propagation to send changes from the capture database to the destination database. If the destination database is the capture database and you are not configuring bi-directional replication, then this propagation between databases is not needed. In this case, the propagation is not configured if the `capture_queue_name` and `apply_queue_name` parameters have the same value. If these values are different, then a propagation is configured between the two queues within the destination database.

Note:

- When these procedures configure downstream capture, they always configure archived-log downstream capture. These procedures do not configure real-time downstream capture. However, you can configure redo transport services for real-time downstream capture before running a procedure, and then set the `downstream_real_time_mine` capture process parameter to `Y` after the procedure completes. You can also modify the scripts generated by these procedures to configure real-time downstream capture.
- If these procedures configure bi-directional replication, then the capture process for the destination database always is a local capture process. That is, these procedures always configure the capture process for changes made to the destination database to run on the destination database.
- Synchronous capture cannot be configured with the configuration procedures.

See Also:

- "Decide Whether Changes Are Allowed at One Database or at Multiple Databases" on page 1-7 and "One-Way or Bi-Directional Replication" on page 2-10 for more information about bi-directional replication
- "Oracle Streams Components Configured by These Procedures" on page 2-9
- *Oracle Streams Concepts and Administration* for information about local capture and downstream capture

Perform Configuration Actions Directly or With a Script

These procedures can configure the Oracle Streams replication environment directly, or they can generate a script that configures the environment. Using a procedure to configure replication directly is simpler than running a script, and the environment is configured immediately. However, you might choose to generate a script for the following reasons:

- You want to review the actions performed by the procedure before configuring the environment.

- You want to modify the script to customize the configuration.

For example, you might want an apply process to use apply handlers for customized processing of the changes to certain tables before applying these changes. In this case, you can use the procedure to generate a script and modify the script to add the apply handlers.

You also might want to maintain DML changes for several tables, but you might want to maintain DDL changes for a subset of these tables. In this case, you can generate a script by running the `MAINTAIN_TABLES` procedure with the `include_ddl` parameter set to `FALSE`. You can modify the script to maintain DDL changes for the appropriate tables.

The `perform_actions` parameter controls whether the procedure configures the replication environment directly:

- To configure an Oracle Streams replication environment directly when you run one of these procedures, set the `perform_actions` parameter to `TRUE`. The default value for this parameter is `TRUE`.
- To generate a configuration script when you run one of these procedures, set the `perform_actions` parameter to `FALSE`, and use the `script_name` and `script_directory_object` parameters to specify the name and location of the configuration script.

Note: The `script_directory_object` parameter cannot point to an Automatic Storage Management (ASM) disk group.

See Also:

- "Decide Whether the Replication Environment Will Use Apply Handlers" on page 1-9
- "Data Definition Language (DDL) Changes" on page 2-13

Oracle Streams Components Configured by These Procedures

These procedures configure the following Oracle Streams clients:

- These procedures configure a capture process that captures changes to the source database. If bi-directional replication is configured, then these procedures also configure a capture process that captures changes to the destination database.
- If the capture database and the destination database are different databases, then these procedures configure a propagation that sends changes from the capture database to the destination database.
- If the capture database and the destination database are the same database, then the queue names determine whether a propagation is created:
  - If the `capture_queue_name` and `apply_queue_name` parameters specify different queue names, then a propagation is created between the two queues within the destination database.
  - If the `capture_queue_name` and `apply_queue_name` parameters specify the same queue name, then a propagation is not created, and the downstream capture process and the apply process use the same queue. This configuration is possible only if the `bi_directional` parameter is set to `FALSE` to configure a single source replication environment.

- If bi-directional replication is configured, then these procedures configure a propagation that sends changes from the destination database to the source database.
- These procedures configure an apply process that applies changes at the destination database. These changes originated at the source database. If bi-directional replication is configured, then these procedures also configure an apply process that applies changes to the source database. These changes originated at the destination database.

By default, the `capture_queue_name` and `apply_queue_name` parameters are set to `NULL`. When these parameters are set to `NULL`, these procedures configure a separate queue for each capture process and apply process. The Oracle Streams replication environment might operate more efficiently if each Oracle Streams client has its own separate queue.

However, two Oracle Streams clients share a queue in the following configurations:

- The configuration described previously in this section in which the downstream capture process and the apply process at the destination database share a queue.
- A configuration in which all of the following conditions are met:
  - The capture database is the source database or a third database.
  - The `bi_directional` parameter is set to `TRUE`.
  - The same queue name is specified for the `capture_queue_name` and `apply_queue_name` parameters.

  In this case, the local capture process and the apply process at the destination database share the same queue. If the source database is the capture database, then the local capture process and the apply process at the source database also share the same queue.

Also, the `capture_name` and `capture_queue_name` parameters must be set to `NULL` when both of the following conditions are met:

- The destination database is the capture database.
- The `bi_directional` parameter is set to `TRUE`.

When both of these conditions are met, these procedure configure two capture processes at the destination database, and these capture processes must have different names. One capture process is the downstream capture process for the source database, while the other capture process is the local capture process that captures changes made to the destination database. When the `capture_name` and `capture_queue_name` parameters are set to `NULL`, the system generates a different name for the capture processes. These procedures raise an error if both conditions are met and either the `capture_name` parameter or the `capture_queue_name` parameter is set to a non-`NULL` value.

See Also:

- "One-Way or Bi-Directional Replication" on page 2-10
- "Local or Downstream Capture for the Source Database" on page 2-7

One-Way or Bi-Directional Replication

These procedures set up either a one-way (or single-source) Oracle Streams configuration with the database specified in the `source_database` parameter as the

source database, or a bi-directional Oracle Streams configuration with both databases acting as source and destination databases. See "Decide Whether Changes Are Allowed at One Database or at Multiple Databases" on page 1-7 for more information.

The `bi_directional` parameter in each procedure controls whether the Oracle Streams configuration is single source or bi-directional:

- If the `bi_directional` parameter is `FALSE`, then a capture process captures changes made to the source database and an apply process at the destination database applies these changes. If the destination database is not the capture database, then a propagation sends the captured changes to the destination database. The default value for this parameter is `FALSE`.
- If the `bi_directional` parameter is `TRUE`, then a separate capture process captures changes made to each database, propagations send these changes to the other database, and each database applies changes from the other database.

When a replication environment is not bi-directional, and no changes are allowed at the destination database, Oracle Streams keeps the shared database objects synchronized at the databases. However, when a replication environment is not bi-directional, and independent changes are allowed at the destination database, the shared database objects might diverge between the databases. Independent changes can be made by users, by applications, or by replication with a third database.

Note:

- You might need to configure conflict resolution if bi-directional replication is configured.
- If you set the `bi_directional` parameter to `TRUE` when you run one of these procedures, then do not allow data manipulation language (DML) or data definition language (DDL) changes to the shared database objects at the destination database while the procedure, or the script generated by the procedure, is running. This restriction does not apply if a procedure is configuring a single-source replication environment.
- These procedures do not configure the replicated tables to be read-only at the destination database. If you set the `bi_directional` parameter to `FALSE` when you run one of these procedures, and the replicated tables should be read only at the destination database, then configure privileges at the destination databases accordingly. However, the apply user for the apply process must be able to make DML changes to the replicated database objects. See *Oracle Database Security Guide* for information about configuring privileges.

See Also:

- Chapter 9, "Oracle Streams Conflict Resolution"
- "Oracle Streams Components Configured by These Procedures" on page 2-9

Change Cycling and Tags Change cycling happens when a change is sent 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.

If the `bi_directional` parameter is set to `TRUE`, then these procedures configure bi-directional replication. To prevent change cycling in a bi-directional Oracle Streams replication environment, these procedures configure the environment in the following way:

- The apply process at each database applies each change with an apply tag that is unique to the environment. An apply tag is an Oracle Streams tag that is part of each redo record created by the apply process. For example, if a procedure configures databases `sfdb.net` and `nydb.net` for bi-directional replication, then the apply tag for the apply process at `sfdb.net` might be the hexidecimal equivalent of `'1'`, and the apply tag for the apply process at `nydb.net` might be the hexidecimal equivalent of `'2'`. In this case, an applied change with a tag that is the hexidecimal equivalent of `'1'` originated at the `nydb.net` database, while an applied change with a tag that is the hexidecimal equivalent of `'2'` originated at the `sfdb.net` database.
- The capture process at each database captures all changes to the shared database objects, regardless of tags in the redo records for the changes to these database objects.
- Each propagation sends all changes made to the shared database objects to the other database in the bi-directional replication environment, except for changes that originated at the other database. Continuing the example, the propagation at `sfdb.net` sends all changes to `nydb.net`, except for changes with a tag value that is the hexidecimal equivalent of `'1'`, because these changes originated at `nydb.net`. Similarly, the propagation at `nydb.net` sends all changes to `sfdb.net`, except for changes with a tag value that is the hexidecimal equivalent of `'2'`. A change that is not propagated because of its tag value is discarded.

These procedures cannot be used to configure multi-directional replication where changes can be cycled back to a source database by a third database in the environment. For example, these procedures cannot be used to configure an Oracle Streams replication environment with three databases where each database shares changes with the other two databases in the environment. Such an environment is sometimes called an "n-way" replication environment. If these procedures were used to configure this type of a three way replication environment, then changes made at a source database would be cycled back to the same source database. In a valid three way replication environment, a particular change is made only once at each database.

These procedures can configure an Oracle Streams replication environment that includes more than two databases, if changes made at a source database cannot cycle back to the same source database. For example, a procedure can be run multiple times to configure an environment in which a primary database shares changes with multiple secondary databases. Such an environment is sometimes called a "hub-and-spoke" replication environment.

You can configure the Oracle Streams environment manually to replicate changes in a multiple source environment where each source database shares changes with the other source databases, or you can modify generated scripts to achieve this.

See Also:

- Chapter 10, "Oracle Streams Tags"
- *Oracle Database 2 Day + Data Replication and Integration Guide* for examples that configure hub-and-spoke replication environments
- *Oracle Streams Extended Examples*

Data Definition Language (DDL) Changes

The `include_ddl` parameter controls whether the procedure configures Oracle Streams replication to maintain DDL changes:

- To configure an Oracle Streams replication environment that does not maintain DDL changes, set the `include_ddl` parameter to `FALSE` when you run one of these procedures. The default value for this parameter is `FALSE`.
- To configure an Oracle Streams replication environment that maintains DDL changes, set the `include_ddl` parameter to `TRUE` when you run one of these procedures.

Note: The `MAINTAIN_SIMPLE_TTS` procedure does not include the `include_ddl` parameter. An Oracle Streams replication environment configured by the `MAINTAIN_SIMPLE_TTS` procedure only maintains DML changes.

See Also: "Decide Whether to Maintain DDL Changes" on page 1-10

Instantiation

The `MAINTAIN_GLOBAL`, `MAINTAIN_SCHEMAS`, and `MAINTAIN_TABLES` procedures provide options for instantiation. Instantiation is the process of preparing database objects for instantiation at a source database, optionally copying the database objects from a source database to a destination database, and setting the instantiation SCN for each instantiated database object.

When you run one of these three procedures, you can choose to perform the instantiation in one of the following ways:

- **Data Pump Export Dump File Instantiation:** This option performs a Data Pump export of the shared database objects at the source database and a Data Pump import of the export dump file at the destination database. The instantiation SCN is set for each shared database object during import.

  To specify this instantiation option, set the `instantiation` parameter to one of the following values:

  - `DBMS_STREAMS_ADM.INSTANTIATION_FULL` if you run the `MAINTAIN_GLOBAL` procedure
  - `DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA` if you run the `MAINTAIN_SCHEMAS` procedure
  - `DBMS_STREAMS_ADM.INSTANTIATION_TABLE` if you run the `MAINTAIN_TABLES` procedure

  If the `bi_directional` parameter is set to `TRUE`, then the procedure also sets the instantiation SCN for each shared database object at the source database.

When you use this option, you must create directory objects to hold the Data Pump files. See "Creating the Required Directory Objects" on page 2-15.

- **Data Pump Network Import Instantiation:** This option performs a network Data Pump import of the shared database objects. A network import means that Data Pump performs the import without using an export dump file. Therefore, directory objects do not need to be created for instantiation purposes when you use this option. The instantiation SCN is set for each shared database object during import.

  To specify this instantiation option, set the `instantiation` parameter to one of the following values:

  - `DBMS_STREAMS_ADM.INSTANTIATION_FULL_NETWORK` if you run the `MAINTAIN_GLOBAL` procedure
  - `DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA_NETWORK` if you run the `MAINTAIN_SCHEMAS` procedure
  - `DBMS_STREAMS_ADM.INSTANTIATION_TABLE_NETWORK` if you run the `MAINTAIN_TABLES` procedure

  If the `bi_directional` parameter is set to `TRUE`, then the procedure also sets the instantiation SCN for each shared database object at the source database.

- **Generate a Configuration Script with No Instantiation Specified:** This option does not perform an instantiation. This setting is valid only if the `perform_actions` parameter is set to `FALSE`, and the procedure generates a configuration script. In this case, the configuration script does not perform an instantiation and does not set the instantiation SCN for each shared database object. Instead, you must perform the instantiation and ensure that instantiation SCN values are set properly.

  To specify this instantiation option, set the `instantiation` parameter to `DBMS_STREAMS_ADM.INSTANTIATION_NONE` in each procedure.

When one of these procedures performs a table instantiation, the tablespace that contains the table must exist at the destination database. When one of these procedures performs a schema instantiation, the tablespace that contains the schema must exist at the destination database.

When these procedures perform a dump file or network instantiation and an instantiated database object does not exist at the destination database, the database object is imported at the destination database, including its supplemental logging specifications from the source database and its supporting database objects, such as indexes and triggers. However, if the database object already exists at the destination database before instantiation, then it is not imported at the destination database. Therefore, the supplemental logging specifications from the source database are not specified for the database object at the destination database, and the supporting database objects are not imported.

The `PRE_INSTANTIATION_SETUP` and `POST_INSTANTIATION_SETUP` procedures do not perform an instantiation. You must perform any required instantiation actions manually after running `PRE_INSTANTIATION_SETUP` and before running `POST_INSTANTIATION_SETUP`. You also must perform any required instantiation actions manually if you use the `MAINTAIN_GLOBAL`, `MAINTAIN_SCHEMAS`, and `MAINTAIN_TABLES` procedures and set the `instantiation` parameter to `DBMS_STREAMS_ADM.INSTANTIATION_NONE`.

In these cases, you can use any instantiation method. For example, you can use Recovery Manager (RMAN) to perform a database instantiation using the RMAN

DUPLICATE or CONVERT DATABASE command or a tablespace instantiation using the RMAN TRANSPORT TABLESPACE command. If the bi_directional parameter is set to TRUE, then ensure that the instantiation SCN values are set properly at the source database and the destination database.

Note:

- The MAINTAIN_SIMPLE_TTS and MAINTAIN_TTS procedures do not provide these instantiation options. These procedures always perform an instantiation by cloning the tablespace or tablespace set, transferring the files required for instantiation to the destination database, and attaching the tablespace or tablespace set at the destination database.
- If one of these procedures performs an instantiation, then the database objects, tablespace, or tablespaces set being configured for replication must exist at the source database.
- If the RMAN DUPLICATE or CONVERT DATABASE command is used for database instantiation, then the destination database cannot be the capture database.
- When the MAINTAIN_TABLES procedure performs a dump file or network instantiation and the instantiated table already exist at the destination database before instantiation, the procedure does not set the instantiation SCN for the table. In this case, you must set the instantiation SCN for the table manually after the procedure completes.

See Also:

- Chapter 8, "Instantiation and Oracle Streams Replication"
- "Perform Configuration Actions Directly or With a Script" on page 2-8
- "One-Way or Bi-Directional Replication" on page 2-10

Creating the Required Directory Objects

A directory object is similar to an alias for a directory on a file system. The following directory objects might be required when you run one of these procedures:

- A script directory object is required if you decided to generate a configuration script. The configuration script is placed in this directory on the computer system where the procedure is run. Use the script_directory_object parameter when you run one of these procedures to specify the script directory object.
- A source directory object is required if you decided to perform a Data Pump export dump file instantiation, and you will use one of the following procedures: MAINTAIN_GLOBAL, MAINTAIN_SCHEMAS, MAINTAIN_SIMPLE_TTS, MAINTAIN_TABLES, or MAINTAIN_TTS. The Data Pump export dump file and log file are placed in this directory on the computer system running the source database. Use the source_directory_object parameter when you run one of these procedures to specify the source directory object. This directory object is not required if you will use the PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP procedures.

- A destination directory object is required if you decided to perform a Data Pump export dump file instantiation, and you will use one of the following procedures: MAINTAIN_GLOBAL, MAINTAIN_SCHEMAS, MAINTAIN_SIMPLE_TTS, MAINTAIN_TABLES, or MAINTAIN_TTS. The Data Pump export dump file is transferred from the computer system running the source database to the computer system running the destination database and placed in this directory on the computer system running the destination database. Use the destination_directory_object parameter when you run one of these procedures to specify the destination directory object. This directory object is not required if you will use the PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP procedures.

Each directory object must be created using the SQL statement CREATE DIRECTORY, and the user who invokes one of the procedures must have READ and WRITE privilege on each directory object.

For example, complete the following steps to create a directory object named db_dir that corresponds to the /usr/db_files directory:

1. In SQL*Plus, connect to the database as the Oracle Streams administrator.

   See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.

2. Create the directory object:

```
CREATE DIRECTORY db_dir AS '/usr/db_files';
```

The user who creates the directory object automatically has READ and WRITE privilege on the directory object. When you are configuring an Oracle Streams replication environment, typically the Oracle Streams administrator creates the directory objects.

Note: The directory objects cannot point to an Automatic Storage Management (ASM) disk group.

See Also:

- "Perform Configuration Actions Directly or With a Script" on page 2-8
- "Instantiation" on page 2-13

Examples That Configure Two-Database Replication with Local Capture

Each of the following examples configures a two-database replication environment that uses one or more local capture processes:

- Configuring Two-Database Global Replication with Local Capture
- Configuring Two-Database Schema Replication with Local Capture
- Configuring Two-Database Table Replication with Local Capture

Configuring Two-Database Global Replication with Local Capture

You can use the following procedures in the DBMS_STREAMS_ADM package to configure replication at the database level:

- MAINTAIN_GLOBAL
- PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP

The MAINTAIN_GLOBAL procedure automatically excludes database objects that are not supported by Oracle Streams from the replication environment. The PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP procedures do not automatically exclude database objects. Instead, these procedures enable you to specify which database objects to exclude from the replication environment. Query the DBA_STREAMS_UNSUPPORTED data dictionary view to determine which database objects are not supported by Oracle Streams. If unsupported database objects are not excluded, then capture errors will result.

The following table lists the decisions that were made about the Oracle Streams replication environment configured in this example.

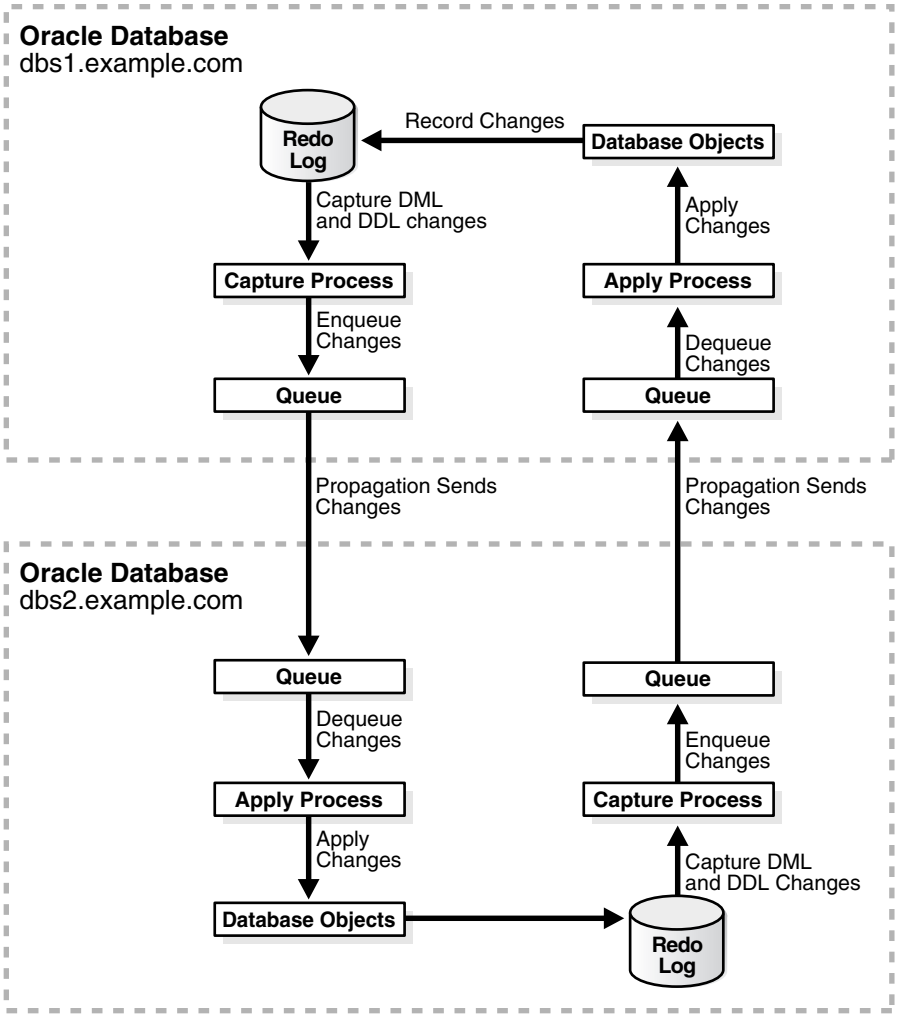
Decision	Assumption for This Example
Decide Which Type of Replication Environment to Configure	This example configures bi-directional replication in a two database environment where both databases are read/write.
Decide Whether to Configure Local or Downstream Capture for the Source Database	This example configures local capture for each source database.
Decide Whether Changes Are Allowed at One Database or at Multiple Databases	This example allows changes to the replicate database objects at both databases.
Decide Whether the Replication Environment Will Have Nonidentical Replicas	This example configures identical shared database objects at the databases.
Decide Whether the Replication Environment Will Use Apply Handlers	This example does not configure apply handlers.
Decide Whether to Maintain DDL Changes	This example maintains DDL changes.
Decide How to Configure the Replication Environment	This example uses the PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP procedures to configure the environment.

In this example, the procedures will configure the replication environment directly. Configuration scripts will not be generated. An RMAN database instantiation will be performed.

As noted in the previous table, this example uses the PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP procedures to configure database replication. The replication configuration will exclude all database objects that are not supported by Oracle Streams. In this example, the source database is dbs1.example.com, and the destination database is dbs2.example.com.

Figure 2–1 provides an overview of the replication environment created in this example.

Figure 2–1 Sample Oracle Streams Environment That Replicates an Entire Database



Note: A capture process never captures changes in the `SYS`, `SYSTEM`, or `CTXSYS` schemas. Changes to these schemas are not maintained by Oracle Streams in the replication configuration described in this section.

See Also: *Oracle Streams Concepts and Administration* for instructions on determining which database objects are not supported by Oracle Streams

Complete the following steps to use the `PRE_INSTANTIATION_SETUP` and `POST_INSTANTIATION_SETUP` procedures to configure the replication environment:

1. Complete the required tasks before running the `PRE_INSTANTIATION_SETUP` procedure. See "Tasks to Complete Before Configuring Oracle Streams Replication" on page 1-13 for instructions.

   For this configuration, the following tasks must be completed:

   - Configure an Oracle Streams administrator at both databases. See "Configuring an Oracle Streams Administrator on All Databases" on page 1-13.

- Configure network connectivity and database links:
  - Configure network connectivity between the source database `dbs1.example.com` and the destination database `dbs2.example.com`.
  - Create a database link from the source database `dbs1.example.com` to the destination database `dbs2.example.com`.

  See "Configuring Network Connectivity and Database Links" on page 1-17.
- Ensure that both databases are in `ARCHIVELOG` mode. See "Ensuring That Each Source Database Is In ARCHIVELOG Mode" on page 1-19.
- Ensure that the initialization parameters are set properly at both databases. See "Setting Initialization Parameters Relevant to Oracle Streams" on page 1-19.
- Configure the Oracle Streams pool properly at both databases. See "Configuring the Oracle Streams Pool" on page 1-25.

A database link is required from the destination database to the source database. However, because RMAN will be used for database instantiation, this database link must be created after instantiation. This database link is required because the replication environment will be bi-directional and because RMAN will be used for database instantiation.

2. In SQL*Plus, connect to the source database `dbs1.example.com` as the Oracle Streams administrator.

   See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.

3. Run the `PRE_INSTANTIATION_SETUP` procedure:

```
DECLARE
  empty_tbs  DBMS_STREAMS_TABLESPACE_ADM.TABLESPACE_SET;
BEGIN
  DBMS_STREAMS_ADM.PRE_INSTANTIATION_SETUP(
    maintain_mode        => 'GLOBAL',
    tablespace_names     => empty_tbs,
    source_database      => 'dbs1.example.com',
    destination_database => 'dbs2.example.com',
    perform_actions      => TRUE,
    bi_directional       => TRUE,
    include_ddl          => TRUE,
    start_processes      => TRUE,
    exclude_schemas      => '*',
    exclude_flags        => DBMS_STREAMS_ADM.EXCLUDE_FLAGS_UNSUPPORTED +
                            DBMS_STREAMS_ADM.EXCLUDE_FLAGS_DML +
                            DBMS_STREAMS_ADM.EXCLUDE_FLAGS_DDL);
END;
/
```

   Notice that the `start_processes` parameter is set to `TRUE`. Therefore, each capture process and apply process created during the configuration is started automatically.

   Also, notice the values specified for the `exclude_schemas` and `exclude_flags` parameters. The asterisk (`*`) specified for `exclude_schemas` indicates that certain database objects in every schema in the database might be excluded from the replication environment. The value specified for the `exclude_flags` parameter indicates that DML and DDL changes for all unsupported database

objects are excluded from the replication environment. Rules are placed in the negative rule sets for the capture processes to exclude these database objects.

Because the procedure is run at the source database, local capture is configured at the source database.

Because this procedure configures a bi-directional replication environment, do not allow DML or DDL changes to the shared database objects at the destination database while the procedure is running.

The procedure does not specify the `apply_name` parameter. Therefore, the default, `NULL`, is specified for this parameter. When the `apply_name` parameter is set to `NULL`, no apply process that applies changes from the source database can exist on the destination database. If an apply process that applies changes from the source database exists at the destination database, then specify a non-`NULL` value for the `apply_name` parameter.

To monitor the progress of the configuration operation, follow the instructions in "Monitoring Oracle Streams Configuration Progress" on page 2-53.

If this procedure encounters an error and stops, then see "Recovering from Operation Errors" on page 12-36 for information about either recovering from the error or rolling back the configuration operation.

4. Perform the instantiation. You can use any of the methods described in Chapter 8, "Instantiation and Oracle Streams Replication" to complete the instantiation. This example uses the RMAN `DUPLICATE` command to perform the instantiation by performing the following steps:

   a. Create a backup of the source database if one does not exist. RMAN requires a valid backup for duplication. In this example, create a backup of `dbs1.example.com` if one does not exist.

   Note: A backup of the source database is not necessary if you use the `FROM ACTIVE DATABASE` option when you run the RMAN `DUPLICATE` command. For large databases, the `FROM ACTIVE DATABASE` option requires significant network resources. This example does not use this option.

   b. In SQL*Plus, connect to the source database `dbs1.example.com` as the Oracle Streams administrator.

      See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.

   c. Determine the until SCN for the RMAN `DUPLICATE` command:

```
SET SERVEROUTPUT ON SIZE 1000000
DECLARE
  until_scn NUMBER;
BEGIN
  until_scn:= DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER;
      DBMS_OUTPUT.PUT_LINE('Until SCN: ' || until_scn);
END;
/
```

      Make a note of the until SCN returned. You will use this number in Step h. For this example, assume that the returned until SCN is `45442631`.

d. In SQL*Plus, connect to the source database `dbs1.example.com` as an administrative user.

e. Archive the current online redo log:

```
ALTER SYSTEM ARCHIVE LOG CURRENT;
```

f. Prepare your environment for database duplication, which includes preparing the destination database as an auxiliary instance for duplication. See *Oracle Database Backup and Recovery User's Guide* for instructions.

g. Start the RMAN client, and connect to the source database `dbs1.example.com` as `TARGET` and to the destination database `dbs2.example.com` as `AUXILIARY`.

See Also: *Oracle Database Backup and Recovery Reference* for more information about the RMAN `CONNECT` command

h. Use the RMAN `DUPLICATE` command with the `OPEN RESTRICTED` option to instantiate the source database at the destination database. The `OPEN RESTRICTED` option is required. This option enables a restricted session in the duplicate database by issuing the following SQL statement: `ALTER SYSTEM ENABLE RESTRICTED SESSION`. RMAN issues this statement immediately before the duplicate database is opened.

You can use the `UNTIL SCN` clause to specify an SCN for the duplication. Use the until SCN determined in Step c for this clause. Archived redo logs must be available for the until SCN specified and for higher SCN values. Therefore, Step e archived the redo log containing the until SCN.

Ensure that you use `TO` *`database_name`* in the `DUPLICATE` command to specify the name of the duplicate database. In this example, the duplicate database is `dbs2.example.com`. Therefore, the `DUPLICATE` command for this example includes `TO dbs2.example.com`.

The following is an example of an RMAN `DUPLICATE` command:

```
RMAN> RUN
      {
        SET UNTIL SCN 45442631;
        ALLOCATE AUXILIARY CHANNEL dbs2 DEVICE TYPE sbt;
        DUPLICATE TARGET DATABASE TO dbs2
        NOFILENAMECHECK
        OPEN RESTRICTED;
      }
```

See Also: *Oracle Database Backup and Recovery Reference* for more information about the RMAN `DUPLICATE` command

i. In SQL*Plus, connect to the destination database as an administrative user.

j. Rename the global name. After an RMAN database instantiation, the destination database has the same global name as the source database. Rename the global name of the destination database back to its original name with the following statement:

```
ALTER DATABASE RENAME GLOBAL_NAME TO dbs2.example.com;
```

k. In SQL*Plus, connect to the destination database `dbs2.example.com` as the Oracle Streams administrator.

l. Drop the database link from the source database to the destination database that was cloned from the source database:

```
DROP DATABASE LINK dbs2.example.com;
```

5. While still connected to the destination database as the Oracle Streams administrator, create a database link from the destination database to the source database:

```
CREATE DATABASE LINK dbs1.example.com CONNECT TO strmadmin
  IDENTIFIED BY password USING 'dbs1.example.com';
```

See Step 1 for information about why this database link is required.

6. In SQL*Plus, connect to the source database `dbs1.example.com` as the Oracle Streams administrator.

7. Run the `POST_INSTANTIATION_SETUP` procedure:

```
DECLARE
  empty_tbs  DBMS_STREAMS_TABLESPACE_ADM.TABLESPACE_SET;
BEGIN
  DBMS_STREAMS_ADM.POST_INSTANTIATION_SETUP(
    maintain_mode        => 'GLOBAL',
    tablespace_names     => empty_tbs,
    source_database      => 'dbs1.example.com',
    destination_database => 'dbs2.example.com',
    perform_actions      => TRUE,
    bi_directional       => TRUE,
    include_ddl          => TRUE,
    start_processes      => TRUE,
    instantiation_scn    => 45442630,
    exclude_schemas      => '*',
    exclude_flags        => DBMS_STREAMS_ADM.EXCLUDE_FLAGS_UNSUPPORTED +
                            DBMS_STREAMS_ADM.EXCLUDE_FLAGS_DML +
                            DBMS_STREAMS_ADM.EXCLUDE_FLAGS_DDL);
END;
/
```

The parameter values specified in both the `PRE_INSTANTIATION_SETUP` and `POST_INSTANTIATION_SETUP` procedures must match, except for the values of the following parameters: `perform_actions`, `script_name`, `script_directory_object`, and `start_processes`.

Also, notice that the `instantiation_scn` parameter is set to `45442630`. The RMAN `DUPLICATE` command duplicates the database up to one less than the SCN value specified in the `UNTIL SCN` clause. Therefore, you should subtract one from the until SCN value that you specified when you ran the `DUPLICATE` command in Step 4h. In this example, the until SCN was set to `45442631`. Therefore, the `instantiation_scn` parameter should be set to 45442631 - 1, or `45442630`.

If the instantiation SCN was set for the shared database objects at the destination database during instantiation, then the `instantiation_scn` parameter should be set to `NULL`. For example, the instantiation SCN might be set during a full database export/import.

Because this procedure configures a bi-directional replication environment, do not allow DML or DDL changes to the shared database objects at the destination database while the procedure is running.

To monitor the progress of the configuration operation, follow the instructions in "Monitoring Oracle Streams Configuration Progress" on page 2-53.

If this procedure encounters an error and stops, then see "Recovering from Operation Errors" on page 12-36 for information about either recovering from the error or rolling back the configuration operation.

8. At the destination database, connect as an administrative user in SQL*Plus and use the `ALTER SYSTEM` statement to disable the `RESTRICTED SESSION`:

```
ALTER SYSTEM DISABLE RESTRICTED SESSION;
```

9. Configure conflict resolution for the shared database objects if necessary.

   Typically, conflicts are possible in a bi-directional replication environment. If conflicts are possible in the environment created by the `PRE_INSTANTIATION_SETUP` and `POST_INSTANTIATION_SETUP` procedures, then configure conflict resolution before you allow users to make changes to the shared database objects.

   See Chapter 9, "Oracle Streams Conflict Resolution" for more information.

The bi-directional replication environment configured in this example has the following characteristics:

- Database supplemental logging is configured at both databases.
- The `dbs1.example.com` database has two queues and queue tables with system-generated names. One queue is for the local capture process, and one queue is for the apply process.
- The `dbs2.example.com` database has two queues and queue tables with system-generated names. One queue is for the local capture process, and one queue is for the apply process.
- At the `dbs1.example.com` database, a capture process with a system-generated name captures DML and DDL changes to all of the database objects in the database that are supported by Oracle Streams.
- At the `dbs2.example.com` database, a capture process with a system-generated name captures DML and DDL changes to all of the database objects in the database that are supported by Oracle Streams.
- A propagation running on the `dbs1.example.com` database with a system-generated name sends the captured changes from a queue at the `dbs1.example.com` database to a queue at the `dbs2.example.com` database.
- A propagation running on the `dbs2.example.com` database with a system-generated name sends the captured changes from a queue at the `dbs2.example.com` database to a queue at the `dbs1.example.com` database.
- At the `dbs1.example.com` database, an apply process with a system-generated name dequeues the changes from its queue and applies them to the database objects.
- At the `dbs2.example.com` database, an apply process with a system-generated name dequeues the changes from its queue and applies them to the database objects.
- Tags are used to avoid change cycling. Specifically, each apply process uses an apply tag so that redo records for changes applied by the apply process include the tag. Each apply process uses an apply tag that is unique in the replication environment. Each propagation discards changes that have the tag of the apply

process running on the same database. See "Change Cycling and Tags" on page 2-11 for more information.

Configuring Two-Database Schema Replication with Local Capture

This example configures an Oracle Streams replication environment that replicates data manipulation language (DML) changes to all of the tables in the `hr` schema. This example configures a two-database replication environment with local capture processes to capture changes. This example uses the global database names `db1.example.com` and `db2.example.com`. However, you can substitute databases in your environment to complete the example.

The following table lists the decisions that were made about the Oracle Streams replication environment configured in this example.

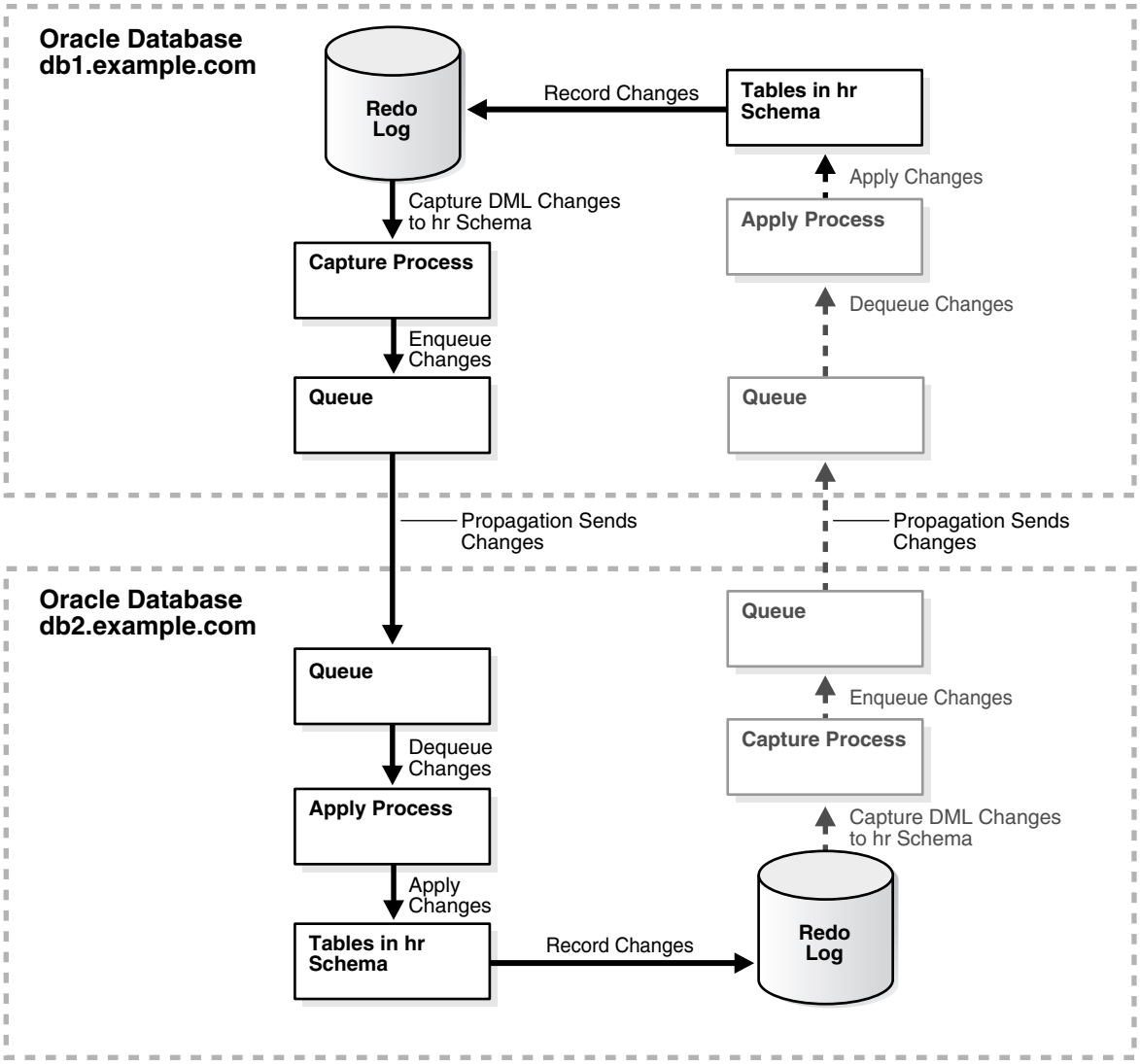
Decision	Assumption for This Example
Decide Which Type of Replication Environment to Configure	This example provides instructions for configuring either one-way or bi-directional replication. To configure bi-directional replication, you must complete additional steps and set the `bi_directional` parameter to `TRUE` when you run the configuration procedure.
Decide Whether to Configure Local or Downstream Capture for the Source Database	This example configures local capture for the source database.
Decide Whether Changes Are Allowed at One Database or at Multiple Databases	This example lets you choose whether to allow changes at one database or both databases.
Decide Whether the Replication Environment Will Have Nonidentical Replicas	This example configures identical shared database objects at the databases.
Decide Whether the Replication Environment Will Use Apply Handlers	This example does not configure apply handlers.
Decide Whether to Maintain DDL Changes	This example maintains DDL changes.
Decide How to Configure the Replication Environment	This example uses the `MAINTAIN_SCHEMAS` procedure to configure the environment.

The database objects being configured for replication might or might not exist at the destination database when you run the `MAINTAIN_SCHEMAS` procedure. If the database objects do not exist at the destination database, then the `MAINTAIN_SCHEMAS` procedure instantiates them at the destination database using a Data Pump export/import. During instantiation, the instantiation SCN is set for these database objects. If the database objects already exist at the destination database, then the `MAINTAIN_SCHEMAS` procedure retains the existing database objects and sets the instantiation SCN for them. In this example, the `hr` schema exists at both the `db1.example.com` database and the `db2.example.com` database before the `MAINTAIN_SCHEMAS` procedure is run.

In this example, the `MAINTAIN_SCHEMAS` procedure will configure the replication environment directly. A configuration script will not be generated. A Data Pump export dump file instantiation will be performed.

Figure 2–2 provides an overview of the environment created in this example. The additional components required for bi-directional replication are shown in gray, and their actions are indicated by dashed lines.

Figure 2–2 Two-Database Replication Environment with Local Capture Processes



Complete the following steps to use the `MAINTAIN_SCHEMAS` procedure to configure the environment:

1. Complete the following tasks to prepare for the two-database replication environment:

   a. Configure network connectivity so that the `db1.example.com` database can communicate with the `db2.example.com` database.

      See *Oracle Database 2 Day DBA* for information about configuring network connectivity between databases.

   b. Configure an Oracle Streams administrator at each database that will participate in the replication environment. See "Configuring an Oracle Streams Administrator on All Databases" on page 1-13 for instructions. This example assumes that the Oracle Streams administrator is `strmadmin`.

   c. Create a database link from the `db1.example.com` database to the `db2.example.com` database.

      The database link should be created in the Oracle Streams administrator's schema. Also, the database link should connect to the Oracle Streams

administrator at the other database. Both the name and the service name of the database link must be `db2.example.com`. See "Configuring Network Connectivity and Database Links" on page 1-17 for instructions.

d. Configure the `db1.example.com` database to run in `ARCHIVELOG` mode. For a capture process to capture changes generated at a source database, the source database must be running in `ARCHIVELOG` mode. See *Oracle Database Administrator's Guide* for information about configuring a database to run in `ARCHIVELOG` mode.

2. To configure a bi-directional replication environment, complete the following steps. If you are configuring a one-way replication environment, then these steps are not required, and you can move on to Step 3.

   a. Create a database link from the `db2.example.com` database to the `db1.example.com` database.

      The database link should be created in the Oracle Streams administrator's schema. Also, the database link should connect to the Oracle Streams administrator at the other database. Both the name and the service name of the database link must be `db1.example.com`. See "Configuring Network Connectivity and Database Links" for instructions.

   b. Configure the `db2.example.com` database to run in `ARCHIVELOG` mode. For a capture process to capture changes generated at a source database, the source database must be running in `ARCHIVELOG` mode. See *Oracle Database Administrator's Guide* for information about configuring a database to run in `ARCHIVELOG` mode.

3. Set initialization parameters properly at each database that will participate in the Oracle Streams replication environment. See "Setting Initialization Parameters Relevant to Oracle Streams""Setting Initialization Parameters Relevant to Oracle Streams" for instructions.

4. Create the following required directory objects:

   - A source directory object at the source database. This example assumes that this directory object is `source_directory`.
   - A destination directory object at the destination database. This example assumes that this directory object is `dest_directory`.

   See "Creating the Required Directory Objects" on page 2-15 for instructions.

5. In SQL*Plus, connect to the `db1.example.com` database as the Oracle Streams administrator.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

6. Run the `MAINTAIN_SCHEMAS` procedure to configure replication of the `hr` schema between the `db1.example.com` database and the `db2.example.com` database.

   Ensure that the `bi_directional` parameter is set properly for the replication environment that you are configuring. Either set this parameter to `FALSE` for one-way replication, or set it to `TRUE` for bi-directional replication.

```
BEGIN
  DBMS_STREAMS_ADM.MAINTAIN_SCHEMAS(
    schema_names                 => 'hr',
    source_directory_object      => 'source_directory',
    destination_directory_object => 'dest_directory',
    source_database              => 'db1.example.com',
```

```
    destination_database         => 'db2.example.com',
    bi_directional               => FALSE); -- Set to TRUE for bi-directional
END;
/
```

The MAINTAIN_SCHEMAS procedure can take some time to run because it is performing many configuration tasks. Do not allow data manipulation language (DML) or data definition language (DDL) changes to the replicated database objects at the destination database while the procedure is running.

When a configuration procedure is run, information about its progress is recorded in the following data dictionary views: DBA_RECOVERABLE_SCRIPT, DBA_RECOVERABLE_SCRIPT_PARAMS, DBA_RECOVERABLE_SCRIPT_BLOCKS, and DBA_RECOVERABLE_SCRIPT_ERRORS. If the procedure stops because it encounters an error, then see *Oracle Streams Replication Administrator's Guide* for instructions about using the RECOVER_OPERATION procedure in the DBMS_STREAMS_ADM package to recover from these errors.

7. If you configured bi-directional replication, then configure latest time conflict resolution for all of the tables in the hr schema at both databases. This schema includes the countries, departments, employees, jobs, job_history, locations, and regions tables. See Chapter 9, "Oracle Streams Conflict Resolution" for instructions.

   See Also: *Oracle Database 2 Day + Data Replication and Integration Guide* for an example that configures this replication environment using Oracle Enterprise Manager

Configuring Two-Database Table Replication with Local Capture

You can use the MAINTAIN_TABLES procedure in the DBMS_STREAMS_ADM package to configure table replication. The example in this section uses this procedure to configure an Oracle Streams replication environment that maintains specific tables in the hr schema. The source database is dbs1.example.com, and the destination database is dbs2.example.com.

The following table lists the decisions that were made about the Oracle Streams replication environment configured in this example.

Decision	Assumption for This Example
Decide Which Type of Replication Environment to Configure	This example configures one-way replication in a two database environment where the source database is read/write and the destination database is read-only.
Decide Whether to Configure Local or Downstream Capture for the Source Database	This example configures local capture for the source database.
Decide Whether Changes Are Allowed at One Database or at Multiple Databases	This example configures a replication environment that allows changes only at the source database.
Decide Whether the Replication Environment Will Have Nonidentical Replicas	This example configures identical shared database objects at the databases.
Decide Whether the Replication Environment Will Use Apply Handlers	This example does not configure apply handlers.

Decision	Assumption for This Example
Decide Whether to Maintain DDL Changes	This example maintains DDL changes for a subset of the shared database objects.
Decide How to Configure the Replication Environment	This example uses the `MAINTAIN_TABLES` procedure to configure the environment.

The replication environment maintains the following DML and DDL changes for the shared database objects:

- The replication environment will maintain DML changes to the following tables in the `hr` schema:
  - `departments`
  - `employees`
  - `countries`
  - `regions`
  - `locations`
  - `jobs`
  - `job_history`
- The replication environment will maintain DDL changes to the following tables in the `hr` schema:
  - `departments`
  - `employees`

The replication environment does not maintain DDL changes to the following tables in the `hr` schema:

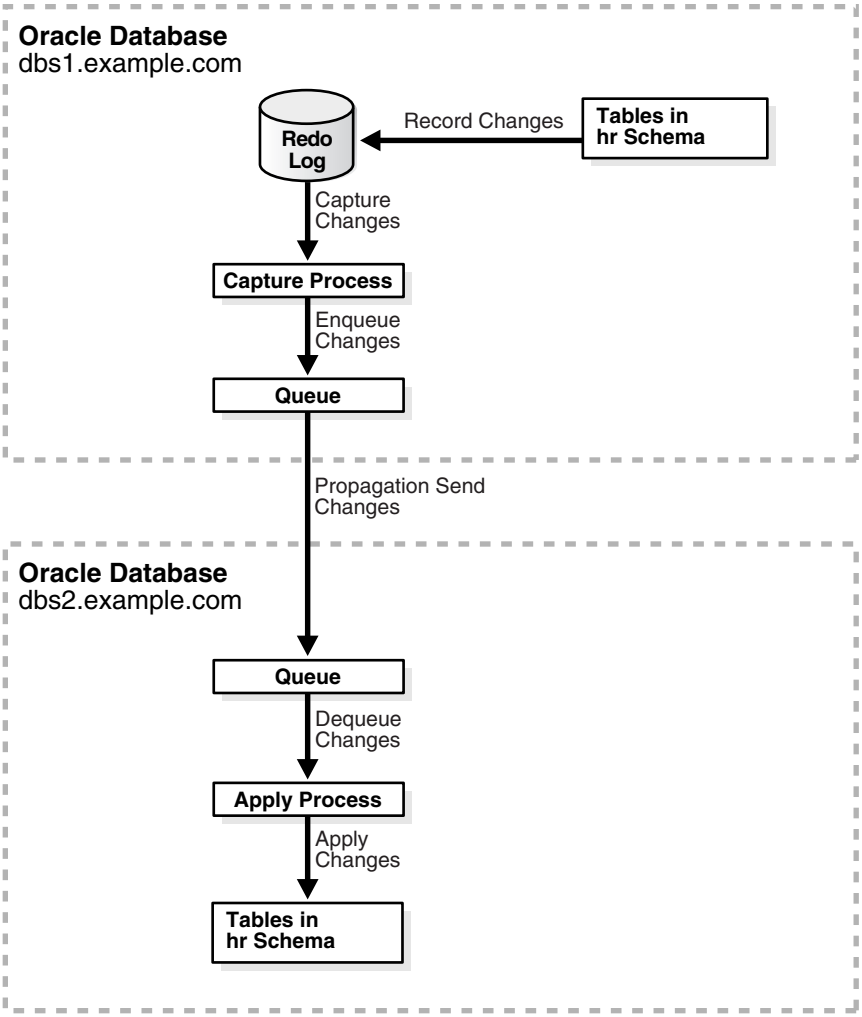
- `countries`
- `regions`
- `locations`
- `jobs`
- `job_history`

In this example, the `MAINTAIN_TABLES` procedure will not configure the replication environment directly. Instead, a configuration script will be generated, and this script will be modified so that DDL changes to the following tables are maintained: `departments` and `employees`. A Data Pump network import instantiation will be performed.

Ensure that you do not try to replicate tables that are not supported by Oracle Streams.

Figure 2–3 provides an overview of the replication environment created in this example.

Figure 2–3 Sample Oracle Streams Environment That Replicates Tables



See Also: *Oracle Streams Concepts and Administration* for instructions on determining which database objects are not supported by Oracle Streams

Complete the following steps to use the `MAINTAIN_TABLES` procedure to configure the environment:

1. Complete the required tasks before running the `MAINTAIN_TABLES` procedure. See "Tasks to Complete Before Configuring Oracle Streams Replication" on page 1-13 for instructions.

   For this configuration, the following tasks must be completed:

   - Configure an Oracle Streams administrator at both databases. See "Configuring an Oracle Streams Administrator on All Databases" on page 1-13.
   - Configure network connectivity and database links:
     - Configure network connectivity between the source database `dbs1.example.com` and the destination database `dbs2.example.com`.
     - Create a database link from the source database `dbs1.example.com` to the destination database `dbs2.example.com`.

  - Because the MAINTAIN_TABLES procedure will perform a Data Pump network import instantiation, create a database link from the destination database dbs2.example.com to the source database dbs1.example.com.

  See "Configuring Network Connectivity and Database Links" on page 1-17.

- Ensure that the source database dbs1.example.com is in ARCHIVELOG mode. See "Ensuring That Each Source Database Is In ARCHIVELOG Mode" on page 1-19.
- Ensure that the initialization parameters are set properly at both databases. See "Setting Initialization Parameters Relevant to Oracle Streams" on page 1-19.
- Configure the Oracle Streams pool properly at both databases. See "Configuring the Oracle Streams Pool" on page 1-25.

2. Create a script directory object at the source database. This example assumes that this directory object is script_directory.

   See "Creating the Required Directory Objects" on page 2-15 for instructions.

3. In SQL*Plus, connect to the source database dbs1.example.com as the Oracle Streams administrator.

   See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.

4. Run the MAINTAIN_TABLES procedure:

```
DECLARE
  tables DBMS_UTILITY.UNCL_ARRAY;
  BEGIN
    tables(1) := 'hr.departments';
    tables(2) := 'hr.employees';
    tables(3) := 'hr.countries';
    tables(4) := 'hr.regions';
    tables(5) := 'hr.locations';
    tables(6) := 'hr.jobs';
    tables(7) := 'hr.job_history';
    DBMS_STREAMS_ADM.MAINTAIN_TABLES(
      table_names                 => tables,
      source_directory_object     => NULL,
      destination_directory_object => NULL,
      source_database             => 'dbs1.example.com',
      destination_database        => 'dbs2.example.com',
      perform_actions             => FALSE,
      script_name                 => 'configure_rep.sql',
      script_directory_object     => 'script_directory',
      bi_directional              => FALSE,
      include_ddl                 => FALSE,
      instantiation      => DBMS_STREAMS_ADM.INSTANTIATION_TABLE_NETWORK);
END;
/
```

   The configure_rep.sql script generated by the procedure uses default values for the parameters that are not specified in the procedure call. The script uses system-generated names for the ANYDATA queues, queue tables, capture process, propagation, and apply process it creates. You can specify different names by using additional parameters available in the MAINTAIN_TABLES procedure. Notice that the include_ddl parameter is set to FALSE. Therefore, the script

does not configure the replication environment to maintain DDL changes to the tables.

The procedure does not specify the `apply_name` parameter. Therefore, the default, `NULL`, is specified for this parameter. When the `apply_name` parameter is set to `NULL`, no apply process that applies changes from the source database can exist on the destination database. If an apply process that applies changes from the source database exists at the destination database, then specify a non-`NULL` value for the `apply_name` parameter.

5. Modify the `configure_rep.sql` script:

   a. Navigate to the directory that corresponds with the `script_directory` directory object on the computer system running the source database.

   b. Open the `configure_rep.sql` script in a text editor. Consider making a backup of this script before modifying it.

   c. In the script, find the `ADD_TABLE_RULES` and `ADD_TABLE_PROPAGATION_RULES` procedure calls that create the table rules for the `hr.departments` and `hr.employees` tables. For example, the procedure calls for the capture process look similar to the following:

```
dbms_streams_adm.add_table_rules(
    table_name => '"HR"."DEPARTMENTS"',
    streams_type => 'CAPTURE',
    streams_name => '"DBS1$CAP"',
    queue_name => '"STRMADMIN"."DBS1$CAPQ"',
    include_dml => TRUE,
    include_ddl => FALSE,
    include_tagged_lcr => TRUE,
    source_database => 'DBS1.EXAMPLE.COM',
    inclusion_rule => TRUE,
    and_condition => get_compatible);

dbms_streams_adm.add_table_rules(
    table_name => '"HR"."EMPLOYEES"',
    streams_type => 'CAPTURE',
    streams_name => '"DBS1$CAP"',
    queue_name => '"STRMADMIN"."DBS1$CAPQ"',
    include_dml => TRUE,
    include_ddl => FALSE,
    include_tagged_lcr => TRUE,
    source_database => 'DBS1.EXAMPLE.COM',
    inclusion_rule => TRUE,
    and_condition => get_compatible);
```

   d. In the procedure calls that you found in Step c, change the setting of the `include_ddl` parameter to `TRUE`. For example, the procedure calls for the capture process should look similar to the following after the modification:

```
dbms_streams_adm.add_table_rules(
    table_name => '"HR"."DEPARTMENTS"',
    streams_type => 'CAPTURE',
    streams_name => '"DBS1$CAP"',
    queue_name => '"STRMADMIN"."DBS1$CAPQ"',
    include_dml => TRUE,
    include_ddl => TRUE,
    include_tagged_lcr => TRUE,
    source_database => 'DBS1.EXAMPLE.COM',
    inclusion_rule => TRUE,
    and_condition => get_compatible);
```

```
dbms_streams_adm.add_table_rules(
    table_name => '"HR"."EMPLOYEES"',
    streams_type => 'CAPTURE',
    streams_name => '"DBS1$CAP"',
    queue_name => '"STRMADMIN"."DBS1$CAPQ"',
    include_dml => TRUE,
    include_ddl => TRUE,
    include_tagged_lcr => TRUE,
    source_database => 'DBS1.EXAMPLE.COM',
    inclusion_rule => TRUE,
    and_condition => get_compatible);
```

Remember to change the procedure calls for all capture processes, propagations, and apply processes.

e. Save and close the `configure_rep.sql` script.

6. In SQL*Plus, connect to the source database `dbs1.example.com` as the Oracle Streams administrator.

7. At the source database, run the configuration script:

```
SET ECHO ON
SPOOL configure_rep.out
@configure_rep.sql
```

The script prompts you to supply information about the database names and the Oracle Streams administrators. When this configuration script completes, the Oracle Streams single-source replication environment is configured. The script also starts the queues, capture process, propagations, and apply process.

The resulting single-source replication environment has the following characteristics:

- At the source database, supplemental logging is configured for the shared database objects.
- The source database `dbs1.example.com` has a queue and queue table with system-generated names.
- The destination database `dbs2.example.com` has a queue and queue table with system-generated names.
- At the source database, a capture process with a system-generated name captures DML changes to all of the tables in the `hr` schema and DDL changes to the `hr.departments` and `hr.employees` tables.
- A propagation running on the source database with a system-generated name sends the captured changes from the queue at the source database to the queue at the destination database.
- At the destination database, an apply process with a system-generated name dequeues the changes from the queue and applies them to the tables at the destination database.

Examples That Configure Two-Database Replication with Downstream Capture

Each of the following examples configures a two-database replication environment that uses a downstream capture process:

- Configuring Tablespace Replication with Downstream Capture at Destination
- Configuring Schema Replication with Downstream Capture at Destination

- Configuring Schema Replication with Downstream Capture at Third Database

Configuring Tablespace Replication with Downstream Capture at Destination

You can use the following procedures in the DBMS_STREAMS_ADM package to configure tablespace replication:

- MAINTAIN_SIMPLE_TTS
- MAINTAIN_TTS
- PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP

You can use the MAINTAIN_SIMPLE_TTS procedure to configure Oracle Streams replication for a simple tablespace, and you can use the MAINTAIN_TTS procedure to configure Oracle Streams replication for a set of self-contained tablespaces. These procedures use transportable tablespaces, Data Pump, the DBMS_STREAMS_TABLESPACE_ADM package, and the DBMS_FILE_TRANSFER package to configure the environment.

A self-contained tablespace has no references from the tablespace pointing outside of the tablespace. For example, if an index in the tablespace is for a table in a different tablespace, then the tablespace is not self-contained. A simple tablespace is a self-contained tablespace that uses only one data file. When there are multiple tablespaces in a tablespace set, a self-contained tablespace set has no references from inside the set of tablespaces pointing outside of the set of tablespaces.

These procedures clone the tablespace or tablespaces being configured for replication from the source database to the destination database. The MAINTAIN_SIMPLE_TTS procedure uses the CLONE_SIMPLE_TABLESPACE procedure in the DBMS_STREAMS_TABLESPACE_ADM package, and the MAINTAIN_TTS procedure uses the CLONE_TABLESPACES procedure in the DBMS_STREAMS_TABLESPACE_ADM package. When a tablespace is cloned, it is made read-only automatically until the clone operation is complete.

The example in this section uses the MAINTAIN_TTS procedure to configure an Oracle Streams replication environment that maintains the following tablespaces using Oracle Streams:

- tbs1
- tbs2

The source database is dbs1.example.com, and the destination database is dbs2.example.com.

The following table lists the decisions that were made about the Oracle Streams replication environment configured in this example.

Decision	Assumption for This Example
Decide Which Type of Replication Environment to Configure	This example configures one-way replication in a two database environment where the source database is read/write and the destination database is read-only.
Decide Whether to Configure Local or Downstream Capture for the Source Database	This example configures a downstream capture process running on the destination database (dbs2.example.com) that captures changes made to the source database (dbs1.example.com). The downstream capture process will be an archived-log downstream capture process.

Decision	Assumption for This Example
Decide Whether Changes Are Allowed at One Database or at Multiple Databases	This example configures a replication environment that allows changes only at the source database.
Decide Whether the Replication Environment Will Have Nonidentical Replicas	This example configures identical shared database objects at the databases.
Decide Whether the Replication Environment Will Use Apply Handlers	This example does not configure apply handlers.
Decide Whether to Maintain DDL Changes	This example maintains DDL changes to the tablespaces and the database objects in the tablespaces.
Decide How to Configure the Replication Environment	This example uses the `MAINTAIN_TTS` procedure to configure the environment.

In this example, the `MAINTAIN_TTS` procedure will configure the replication environment directly. A configuration script will not be generated. In addition, this example makes the following assumptions:

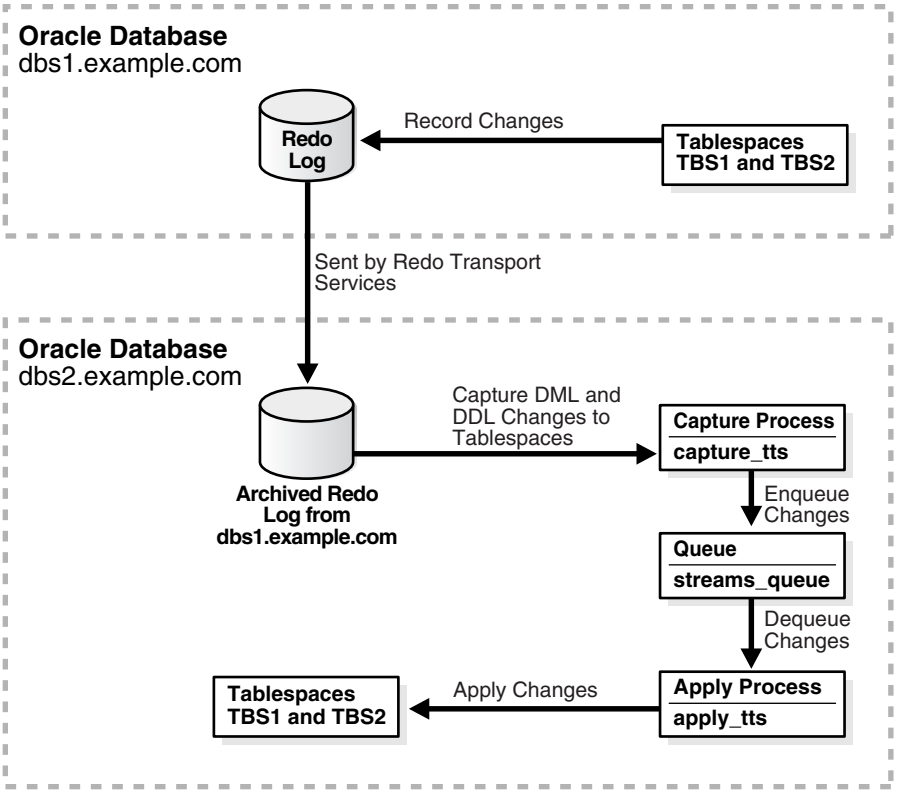
- The tablespaces `tbs1` and `tbs2` make a self-contained tablespace set at the source database `dbs1.example.com`.
- The data files for the tablespace set are both in the `/orc/dbs` directory at the source database `dbs1.example.com`.
- The `dbs2.example.com` database does not contain the tablespace set currently.

The `MAINTAIN_SIMPLE_TTS` and `MAINTAIN_TTS` procedures automatically exclude database objects that are not supported by Oracle Streams from the replication environment by adding rules to the negative rule set of each capture and apply process. The `PRE_INSTANTIATION_SETUP` and `POST_INSTANTIATION_SETUP` procedures enable you to specify which database objects to exclude from the replication environment.

Query the `DBA_STREAMS_UNSUPPORTED` data dictionary view to determine which database objects are not supported by Oracle Streams. If unsupported database objects are not excluded, then capture errors will result.

Figure 2–4 provides an overview of the replication environment created in this example.

Figure 2–4 Sample Oracle Streams Environment That Replicates Tablespaces



> **See Also:** *Oracle Streams Concepts and Administration* for instructions on determining which database objects are not supported by Oracle Streams

Complete the following steps to use the `MAINTAIN_TTS` procedure to configure the environment:

1. Complete the required tasks before running the `MAINTAIN_TTS` procedure. See "Tasks to Complete Before Configuring Oracle Streams Replication" on page 1-13 for instructions.

   For this configuration, the following tasks must be completed:

   - Configure an Oracle Streams administrator at both databases. See "Configuring an Oracle Streams Administrator on All Databases" on page 1-13.
   - Configure network connectivity and database links:
     - Configure network connectivity between the source database `dbs1.example.com` and the destination database `dbs2.example.com`.
     - Create a database link from the source database `dbs1.example.com` to the destination database `dbs2.example.com`.
     - Because downstream capture will be configured at the destination database, create a database link from the destination database `dbs2.example.com` to the source database `dbs1.example.com`.

     See "Configuring Network Connectivity and Database Links" on page 1-17.

- Ensure that both databases are in ARCHIVELOG mode. See "Ensuring That Each Source Database Is In ARCHIVELOG Mode" on page 1-19.
- Ensure that the initialization parameters are set properly at both databases. See "Setting Initialization Parameters Relevant to Oracle Streams" on page 1-19.
- Configure the Oracle Streams pool properly at both databases. See "Configuring the Oracle Streams Pool" on page 1-25.
- Because the destination database will be the capture database for changes made to the source database, configure log file copying from the source database dbs1.example.com to the destination database dbs2.example.com. The capture process will be an archived-log downstream capture process. See "Configuring Log File Transfer to a Downstream Capture Database" on page 1-33.

2. Create the following required directory objects:
   - A source directory object at the source database. This example assumes that this directory object is source_directory.
   - A destination directory object at the destination database. This example assumes that this directory object is dest_directory.

   See "Creating the Required Directory Objects" on page 2-15 for instructions.
3. In SQL*Plus, connect to the database that contains the tablespace set as the Oracle Streams administrator. In this example, connect to the dbs1.example.com database.

   See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.
4. Create a directory object for the directory that contains the data files for the tablespaces in the tablespace set. For example, the following statement creates a directory object named tbs_directory that corresponds to the /orc/dbs directory:

```
CREATE DIRECTORY tbs_directory AS '/orc/dbs';
```

   If the data files are in multiple directories, then a directory object must exist for each of these directories, and the user who runs the MAINTAIN_TTS procedure in Step 6 must have READ privilege on these directory objects. In this example, the Oracle Streams administrator has this privilege because this user creates the directory object.
5. In SQL*Plus, connect to the destination database dbs2.example.com as the Oracle Streams administrator.
6. Run the MAINTAIN_TTS procedure:

```
DECLARE
  t_names  DBMS_STREAMS_TABLESPACE_ADM.TABLESPACE_SET;
BEGIN
  -- Tablespace names
  t_names(1) := 'TBS1';
  t_names(2) := 'TBS2';
  DBMS_STREAMS_ADM.MAINTAIN_TTS(
     tablespace_names             => t_names,
     source_directory_object      => 'source_directory',
     destination_directory_object => 'dest_directory',
     source_database              => 'dbs1.example.com',
```

```
    destination_database         => 'dbs2.example.com',
    perform_actions              => TRUE,
    capture_name                 => 'capture_tts',
    capture_queue_table          => 'streams_queue_table',
    capture_queue_name           => 'streams_queue',
    apply_name                   => 'apply_tts',
    apply_queue_table            => 'streams_queue_table',
    apply_queue_name             => 'streams_queue');
    bi_directional               => FALSE,
    include_ddl                  => TRUE);
END;
/
```

When this procedure completes, the Oracle Streams single-source replication environment is configured.

Because the procedure is run at the destination database, downstream capture is configured at the destination database for changes to the source database. When you use a configuration procedure to configure downstream capture, the parameters that specify the queue and queue table names are important. In such a configuration, it is more efficient for the capture process and apply process to use the same queue at the downstream capture database to avoid propagating changes between queues. To improve efficiency in this sample configuration, notice that `streams_queue` is specified for both the `capture_queue_name` and `apply_queue_name` parameters. Also, `streams_queue_table` is specified for both the `capture_queue_table` and `apply_queue_table` parameters.

To monitor the progress of the configuration operation, follow the instructions in "Monitoring Oracle Streams Configuration Progress" on page 2-53.

If this procedure encounters an error and stops, then see "Recovering from Operation Errors" on page 12-36 for information about either recovering from the error or rolling back the configuration operation.

The resulting single-source replication environment has the following characteristics:

- Supplemental logging is configured for the shared database objects at the source database `dbs1.example.com`.
- The `dbs1.example.com` database has a queue named `streams_queue` which uses a queue table named `streams_queue_table`. This queue is for the apply process.
- The `dbs2.example.com` database has a queue named `streams_queue` which uses a queue table named `streams_queue_table`. This queue is shared by the downstream capture process and the apply process.
- At the `dbs2.example.com` database, an archived-log downstream capture process named `capture_tts` captures changes made to the source database. Specifically, this downstream capture process captures DML changes made to the tables in the `tbs1` and `tbs2` tablespaces and DDL changes to these tablespaces and the database objects in them.

  If the capture process is not enabled after an inordinately long time, then check the alert log for errors. See *Oracle Streams Concepts and Administration* for more information.
- At the `dbs2.example.com` database, an apply process named `apply_tts` dequeues the changes from its queue and applies them to the shared database objects.

Configuring Schema Replication with Downstream Capture at Destination

This example configures an Oracle Streams replication environment that replicates data manipulation language (DML) changes to all of the tables in the `hr` schema. This example configures a two-database replication environment with a downstream capture process at the destination database. This example uses the global database names `src.example.com` and `dest.example.com`. However, you can substitute databases in your environment to complete the example. See "Decide Which Type of Replication Environment to Configure" on page 1-4 for more information about two-database replication environments.

In this example, the downstream capture process runs on the destination database `dest.example.com`. Therefore, the resources required to capture changes are freed at the source database `src.example.com`. This example configures a real-time downstream capture process, not an archived-log downstream capture process. The advantage of real-time downstream capture is that it reduces the amount of time required to capture the changes made at the source database. The time is reduced because the real-time downstream capture process does not need to wait for the redo log file to be archived before it can capture data from it.

The following table lists the decisions that were made about the Oracle Streams replication environment configured in this example.

Decision	Assumption for This Example
Decide Which Type of Replication Environment to Configure	This example configures one-way replication in a two database environment where the source database is read/write and the destination database is read-only.
Decide Whether to Configure Local or Downstream Capture for the Source Database	This example configures a downstream capture process running on the destination database (`dest.example.com`) that captures changes made to the source database (`src.example.com`). The downstream capture process will be a real-time downstream capture process.
Decide Whether Changes Are Allowed at One Database or at Multiple Databases	This example configures a replication environment that allows changes only at the source database.
Decide Whether the Replication Environment Will Have Nonidentical Replicas	This example configures identical shared database objects at the databases.
Decide Whether the Replication Environment Will Use Apply Handlers	This example does not configure apply handlers.
Decide Whether to Maintain DDL Changes	This example maintains DDL changes to the tablespaces and the database objects in the tablespaces.
Decide How to Configure the Replication Environment	This example uses the `MAINTAIN_SCHEMAS` procedure to configure the environment.

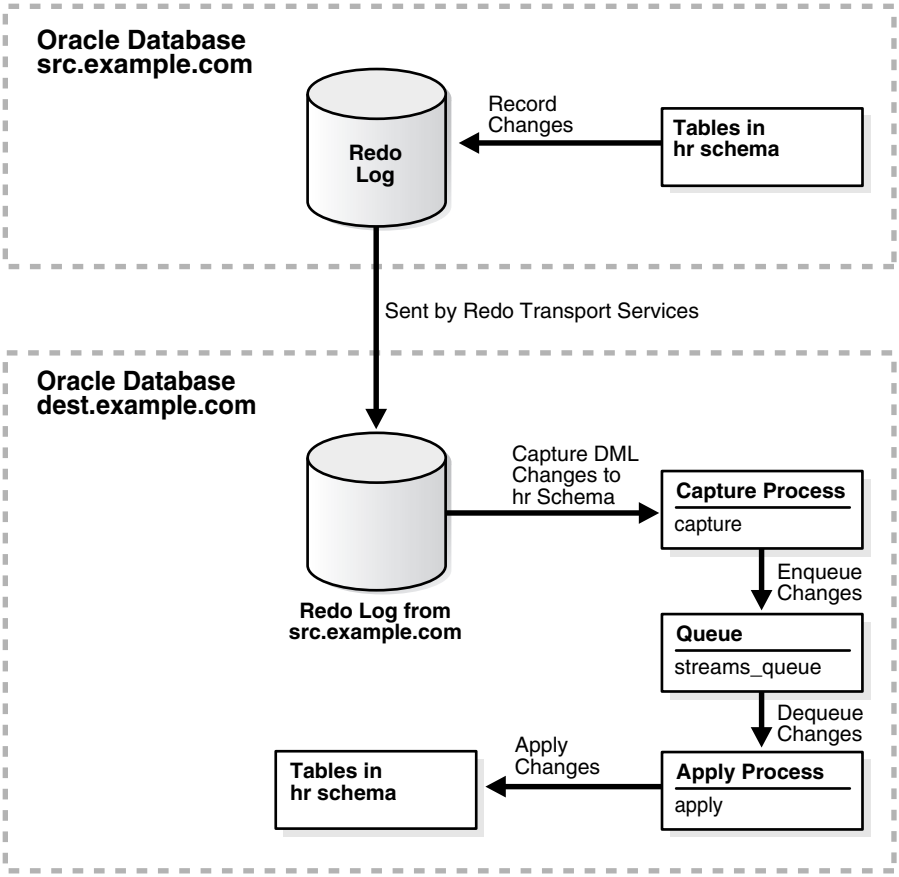
The database objects being configured for replication might or might not exist at the destination database when you run the `MAINTAIN_SCHEMAS` procedure. If the database objects do not exist at the destination database, then the `MAINTAIN_SCHEMAS` procedure instantiates them at the destination database using a Data Pump export/import. During instantiation, the instantiation SCN is set for these database objects. If the database objects already exist at the destination database, then the `MAINTAIN_SCHEMAS` procedure retains the existing database objects and sets the instantiation SCN for them. In this example, the `hr` schema exists at both the

src.example.com database and the dest.example.com database before the MAINTAIN_SCHEMAS procedure is run.

In this example, the MAINTAIN_SCHEMAS procedure will configure the replication environment directly. A configuration script will not be generated. A Data Pump export dump file instantiation will be performed.

Figure 2–5 provides an overview of the environment created in this example.

Figure 2–5 Two-Database Replication Environment with a Downstream Capture Process



Complete the following steps to use the MAINTAIN_SCHEMAS procedure to configure the environment:

1. Complete the following tasks to prepare for the two-database replication environment:

   a. Configure network connectivity so that the src.example.com database and the dest.example.com database can communicate with each other.

      See *Oracle Database 2 Day DBA* for information about configuring network connectivity between databases.

   b. Configure an Oracle Streams administrator at each database that will participate in the replication environment. See "Configuring an Oracle Streams Administrator on All Databases" on page 1-13 for instructions. This example assumes that the Oracle Streams administrator is strmadmin.

c. Create a database link from the source database to the destination database and from the destination database to the source database. In this example, create the following database links:

   – From the `src.example.com` database to the `dest.example.com` database. Both the name and the service name of the database link must be `dest.example.com`.

   – From the `dest.example.com` database to the `src.example.com` database. Both the name and the service name of the database link must be `src.example.com`.

   The database link from the `dest.example.com` database to the `src.example.com` database is necessary because the `src.example.com` database is the source database for the downstream capture process at the `dest.example.com` database. This database link simplifies the creation and configuration of the capture process.

   Each database link should be created in the Oracle Streams administrator's schema. Also, each database link should connect to the Oracle Streams administrator at the other database. See "Configuring Network Connectivity and Database Links" on page 1-17 for instructions.

d. Set initialization parameters properly at each database that will participate in the Oracle Streams replication environment. See "Setting Initialization Parameters Relevant to Oracle Streams" on page 1-19 for instructions.

e. Configure both databases to run in `ARCHIVELOG` mode. For a downstream capture process to capture changes generated at a source database, both the source database and the downstream capture database must be running in `ARCHIVELOG` mode. In this example, the `src.example.com` and `dest.example.com` databases must be running in `ARCHIVELOG` mode. See *Oracle Database Administrator's Guide* for information about configuring a database to run in `ARCHIVELOG` mode.

f. Because the destination database (`dest.example.com`) will be the capture database for changes made to the source database, configure log file copying from the source database `src.example.com` to the destination database `dest.example.com`. See "Configuring Log File Transfer to a Downstream Capture Database" on page 1-33.

g. Because this example configures a real-time downstream capture process, add standby redo logs at the downstream database. See "Adding Standby Redo Logs for Real-Time Downstream Capture" on page 1-36.

2. Create the following required directory objects:

   - A source directory object at the source database. This example assumes that this directory object is `source_directory`.
   - A destination directory object at the destination database. This example assumes that this directory object is `dest_directory`.

   See "Creating the Required Directory Objects" on page 2-15 for instructions.

3. In SQL*Plus, connect to the `dest.example.com` database as the Oracle Streams administrator.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

4. While still connected to the `dest.example.com` database as the Oracle Streams administrator, run the `MAINTAIN_SCHEMAS` procedure to configure replication between the `src.example.com` database and the `dest.example.com` database:

```
BEGIN
  DBMS_STREAMS_ADM.MAINTAIN_SCHEMAS(
    schema_names                 => 'hr',
    source_directory_object      => 'source_directory',
    destination_directory_object => 'dest_directory',
    source_database              => 'src.example.com',
    destination_database         => 'dest.example.com',
    capture_name                 => 'capture',
    capture_queue_table          => 'streams_queue_qt',
    capture_queue_name           => 'streams_queue',
    apply_name                   => 'apply',
    apply_queue_table            => 'streams_queue_qt',
    apply_queue_name             => 'streams_queue');
END;
/
```

   The `MAINTAIN_SCHEMAS` procedure can take some time to run because it is performing many configuration tasks. Do not allow data manipulation language (DML) or data definition language (DDL) changes to the replicated database objects at the destination database while the procedure is running.

   In the `MAINTAIN_SCHEMAS` procedure, only the following parameters are required: `schema_names`, `source_directory_object`, `destination_directory_object`, `source_database`, and `destination_database`.

   This example specifies the other parameters to show that you can choose the name for the capture process, capture process's queue table, capture process's queue, apply process, apply process's queue table, and apply process's queue. If you do not specify these parameters, then system-generated names are used.

   When you use a configuration procedure to configure downstream capture, the parameters that specify the queue and queue table names are important. In such a configuration, it is more efficient for the capture process and apply process to use the same queue at the downstream capture database to avoid propagating changes between queues. To improve efficiency in this sample configuration, notice that `streams_queue` is specified for both the `capture_queue_name` and `apply_queue_name` parameters. Also, `streams_queue_qt` is specified for both the `capture_queue_table` and `apply_queue_table` parameters.

   When a configuration procedure is run, information about its progress is recorded in the following data dictionary views: `DBA_RECOVERABLE_SCRIPT`, `DBA_RECOVERABLE_SCRIPT_PARAMS`, `DBA_RECOVERABLE_SCRIPT_BLOCKS`, and `DBA_RECOVERABLE_SCRIPT_ERRORS`. If the procedure stops because it encounters an error, then see *Oracle Streams Replication Administrator's Guide* for instructions about using the `RECOVER_OPERATION` procedure in the `DBMS_STREAMS_ADM` package to recover from these errors.

   Wait until the procedure completes successfully before proceeding to the next step.

5. While still connected to the `dest.example.com` database as the Oracle Streams administrator, set the `downstream_real_time_mine` capture process parameter to `Y`:

```
BEGIN
  DBMS_CAPTURE_ADM.SET_PARAMETER(
    capture_name => 'capture',
```

```
    parameter    => 'downstream_real_time_mine',
    value        => 'Y');
END;
/
```

6. In SQL*Plus, connect to the source database `src.example.com` as an administrative user.
7. Archive the current log file at the source database:

   ```
   ALTER SYSTEM ARCHIVE LOG CURRENT;
   ```

   Archiving the current log file at the source database starts real-time mining of the source database redo log.

   If the capture process appears to be waiting for redo data for an inordinately long time, then check the alert log for errors. See *Oracle Streams Concepts and Administration* for more information.

   See Also: *Oracle Database 2 Day + Data Replication and Integration Guide* for an example that configures this replication environment using Oracle Enterprise Manager

Configuring Schema Replication with Downstream Capture at Third Database

You can use the `MAINTAIN_SCHEMAS` procedure in the `DBMS_STREAMS_ADM` package to configure schema replication. The example in this section uses this procedure to configure an Oracle Streams replication environment that maintains the `hr` schema. The source database is `dbs1.example.com`, and the destination database is `dbs3.example.com`.

The following table lists the decisions that were made about the Oracle Streams replication environment configured in this example.

Decision	Assumption for This Example
Decide Which Type of Replication Environment to Configure	This example configures bi-directional replication in a two database environment where both databases are read/write.
Decide Whether to Configure Local or Downstream Capture for the Source Database	This example configures a downstream capture process running on a third database named `dbs2.example.com` that captures changes made to the source database (`dbs1.example.com`), and a propagation at `dbs2.example.com` will propagate these captured changes to the destination database (`dbs3.example.com`). The downstream capture process will be a real-time downstream capture process.
Decide Whether Changes Are Allowed at One Database or at Multiple Databases	This example configures a replication environment that allows changes to the replicated database objects at both databases.
Decide Whether the Replication Environment Will Have Nonidentical Replicas	This example configures identical shared database objects at the databases.
Decide Whether the Replication Environment Will Use Apply Handlers	This example does not configure apply handlers.

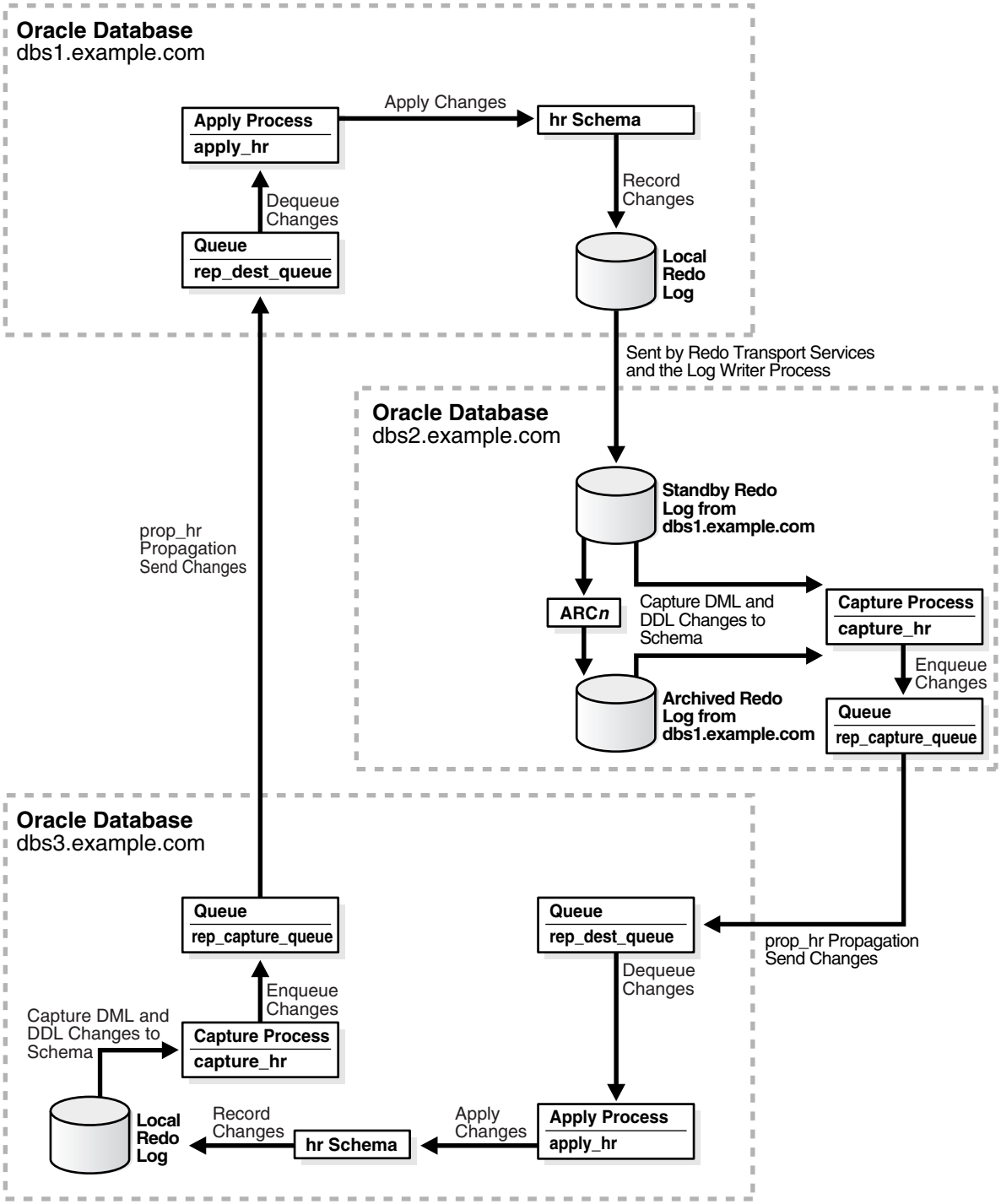
Decision	Assumption for This Example
Decide Whether to Maintain DDL Changes	This example maintains DDL changes to `hr` schema and the database objects in the `hr` schema will be maintained.
Decide How to Configure the Replication Environment	This example uses the `MAINTAIN_SCHEMAS` procedure to configure the environment.

In this example, the `MAINTAIN_SCHEMAS` procedure will configure the replication environment directly. A configuration script will not be generated. A Data Pump export dump file instantiation will be performed.

The `MAINTAIN_SCHEMAS` procedure automatically excludes database objects that are not supported by Oracle Streams from the replication environment by adding rules to the negative rule set of each capture and apply process. Query the `DBA_STREAMS_UNSUPPORTED` data dictionary view to determine which database objects are not supported by Oracle Streams. If unsupported database objects are not excluded, then capture errors will result.

Figure 2–6 provides an overview of the replication environment created in this example.

Figure 2–6 Sample Oracle Streams Environment That Replicates a Schema



See Also: *Oracle Streams Concepts and Administration* for instructions on determining which database objects are not supported by Oracle Streams

Complete the following steps to use the `MAINTAIN_SCHEMAS` procedure to configure the environment:

1. Complete the required tasks before running the `MAINTAIN_SCHEMAS` procedure. See "Tasks to Complete Before Configuring Oracle Streams Replication" on page 1-13 for instructions.

   For this configuration, the following tasks must be completed:

   - Configure an Oracle Streams administrator at all three databases. See "Configuring an Oracle Streams Administrator on All Databases" on page 1-13.
   - Configure network connectivity and database links:
     - Configure network connectivity between all three databases: the source database `dbs1.example.com`, the destination database `dbs3.example.com`, and the third database `dbs2.example.com`.
     - Create a database link from the source database `dbs1.example.com` to the destination database `dbs3.example.com`.
     - Because downstream capture will be configured at the third database, create a database link from the third database `dbs2.example.com` to the source database `dbs1.example.com`.
     - Because downstream capture will be configured at the third database, create a database link from the third database `dbs2.example.com` to the destination database `dbs3.example.com`.
     - Because the replication environment will be bi-directional, create a database link from the destination database `dbs3.example.com` to the source database `dbs1.example.com`.

     See "Configuring Network Connectivity and Database Links" on page 1-17.
   - Ensure that the source database, the destination databases, and the third database are in `ARCHIVELOG` mode. See "Ensuring That Each Source Database Is In ARCHIVELOG Mode" on page 1-19.
   - Ensure that the initialization parameters are set properly at all databases. See "Setting Initialization Parameters Relevant to Oracle Streams" on page 1-19.
   - Configure the Oracle Streams pool properly at both databases. See "Configuring the Oracle Streams Pool" on page 1-25.
   - Because a third database (`dbs2.example.com`) will be the capture database for changes made to the source database, configure log file copying from the source database `dbs1.example.com` to the third database `dbs2.example.com`. See "Configuring Log File Transfer to a Downstream Capture Database" on page 1-33.
   - Because this example configures a real-time downstream capture process, add standby redo logs at the downstream database (`dbs2.example.com`). See "Adding Standby Redo Logs for Real-Time Downstream Capture" on page 1-36.

2. Create the following required directory objects:

   - A source directory object at the source database. This example assumes that this directory object is `source_directory`.
   - A destination directory object at the destination database. This example assumes that this directory object is `dest_directory`.

   See "Creating the Required Directory Objects" on page 2-15 for instructions.

3. In SQL*Plus, connect to the third database `dbs2.example.com` as the Oracle Streams administrator.

   See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.

4. Run the `MAINTAIN_SCHEMAS` procedure:

```
BEGIN
  DBMS_STREAMS_ADM.MAINTAIN_SCHEMAS(
    schema_names                 => 'hr',
    source_directory_object      => 'source_directory',
    destination_directory_object => 'dest_directory',
    source_database              => 'dbs1.example.com',
    destination_database         => 'dbs3.example.com',
    perform_actions              => TRUE,
    dump_file_name               => 'export_hr.dmp',
    capture_queue_table          => 'rep_capture_queue_table',
    capture_queue_name           => 'rep_capture_queue',
    capture_queue_user           => NULL,
    apply_queue_table            => 'rep_dest_queue_table',
    apply_queue_name             => 'rep_dest_queue',
    apply_queue_user             => NULL,
    capture_name                 => 'capture_hr',
    propagation_name             => 'prop_hr',
    apply_name                   => 'apply_hr',
    log_file                     => 'export_hr.clg',
    bi_directional               => TRUE,
    include_ddl                  => TRUE,
    instantiation                => DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA);
END;
/
```

   Because this procedure configures a bi-directional replication environment, do not allow DML or DDL changes to the shared database objects at the destination database while the procedure is running.

   Because the procedure is run at the third database, downstream capture is configured at the third database for changes to the source database.

   To monitor the progress of the configuration operation, follow the instructions in "Monitoring Oracle Streams Configuration Progress" on page 2-53.

   If this procedure encounters an error and stops, then see "Recovering from Operation Errors" on page 12-36 for information about either recovering from the error or rolling back the configuration operation.

5. Set the `downstream_real_time_mine` capture process parameter to `Y`:

```
BEGIN
  DBMS_CAPTURE_ADM.SET_PARAMETER(
    capture_name => 'capture_hr',
    parameter    => 'downstream_real_time_mine',
    value        => 'Y');
END;
/
```

6. Connect to the source database `dbs1.example.com` as an administrative user with the necessary privileges to switch log files.

7. Archive the current log file at the source database:

```
ALTER SYSTEM ARCHIVE LOG CURRENT;
```

Archiving the current log file at the source database starts real time mining of the source database redo log.

If the capture process appears to be waiting for redo data for an inordinately long time, then check the alert log for errors. See *Oracle Streams Concepts and Administration* for more information.

8. Configure conflict resolution for the shared database objects if necessary.

   Typically, conflicts are possible in a bi-directional replication environment. If conflicts are possible in the environment created by the `MAINTAIN_SCHEMAS` procedure, then configure conflict resolution before you allow users to make changes to the shared database objects.

   See Chapter 9, "Oracle Streams Conflict Resolution" for information.

The bi-directional replication environment configured in this example has the following characteristics:

- Supplemental logging is configured for the shared database objects at the source and destination databases.
- The `dbs1.example.com` database has a queue named `rep_dest_queue` which uses a queue table named `rep_dest_queue_table`. This queue is for the apply process.
- The `dbs3.example.com` database has a queue named `rep_capture_queue` which uses a queue table named `rep_capture_queue_table`. This queue is for the local capture process.
- The `dbs3.example.com` database has a queue named `rep_dest_queue` which uses a queue table named `rep_dest_queue_table`. This queue is for the apply process.
- The `dbs2.example.com` database has a queue named `rep_capture_queue` which uses a queue table named `rep_capture_queue_table`. This queue is for the downstream capture process.
- At the `dbs2.example.com` database, a real-time downstream capture process named `capture_hr` captures DML and DDL changes to the `hr` schema and the database objects in the schema at the source database.
- At the `dbs3.example.com` database, a local capture process named `capture_hr` captures DML and DDL changes to the `hr` schema and the database objects in the schema at the destination database.
- A propagation running on the `dbs2.example.com` database named `prop_hr` sends the captured changes from the queue in the `dbs2.example.com` database to the queue in the `dbs3.example.com` database.
- A propagation running on the `dbs3.example.com` database named `prop_hr` sends the captured changes from the queue in the `dbs3.example.com` database to the queue in the `dbs1.example.com` database.
- At the `dbs1.example.com` database, an apply process named `apply_hr` dequeues the changes from `rep_dest_queue` and applies them to the database objects.
- At the `dbs3.example.com` database, an apply process named `apply_hr` dequeues the changes from `rep_dest_queue` and applies them to the database objects.

- Tags are used to avoid change cycling. Specifically, each apply process uses an apply tag so that redo records for changes applied by the apply process include the tag. Each apply process uses an apply tag that is unique in the replication environment. Each propagation discards changes that have the tag of the apply process running on the same database. See "Change Cycling and Tags" on page 2-11 for more information.

Example That Configures Hub-and-Spoke Replication

This example configures an Oracle Streams hub-and-spoke replication environment that replicates data manipulation language (DML) changes to all of the tables in the `hr` schema. This example uses a capture process at each database to capture these changes. Hub-and-spoke replication means that a central hub database replicates changes with one or more spoke databases. The spoke databases do not communicate with each other directly. In this sample configuration, the hub database sends changes generated at one spoke database to the other spoke database.

In this example, the global name of the hub database is `hub.example.com`, and the global names of the spoke databases are `spoke1.example.com` and `spoke2.example.com`. However, you can substitute databases in your environment to complete the example.

The following table lists the decisions that were made about the Oracle Streams replication environment configured in this example.

Decision	Assumption for This Example
Decide Which Type of Replication Environment to Configure	This example configures a hub-and-spoke replication environment in which the global name of the hub database is `hub.example.com`, and the global names of the spoke databases are `spoke1.example.com` and `spoke2.example.com`. All of the databases in the environment are read/write.
Decide Whether to Configure Local or Downstream Capture for the Source Database	This example configures local capture at each database.
Decide Whether Changes Are Allowed at One Database or at Multiple Databases	This example configures a replication environment that allows changes to the replicated database objects at all three databases.
Decide Whether the Replication Environment Will Have Nonidentical Replicas	This example configures identical shared database objects at the databases.
Decide Whether the Replication Environment Will Use Apply Handlers	This example does not configure apply handlers.
Decide Whether to Maintain DDL Changes	This example does not maintain DDL changes to the shared database objects.
Decide How to Configure the Replication Environment	This example uses the `MAINTAIN_SCHEMAS` procedure to configure the environment.

In this example, the `MAINTAIN_SCHEMAS` procedure will configure the replication environment directly. A configuration script will not be generated. A Data Pump export dump file instantiation will be performed.

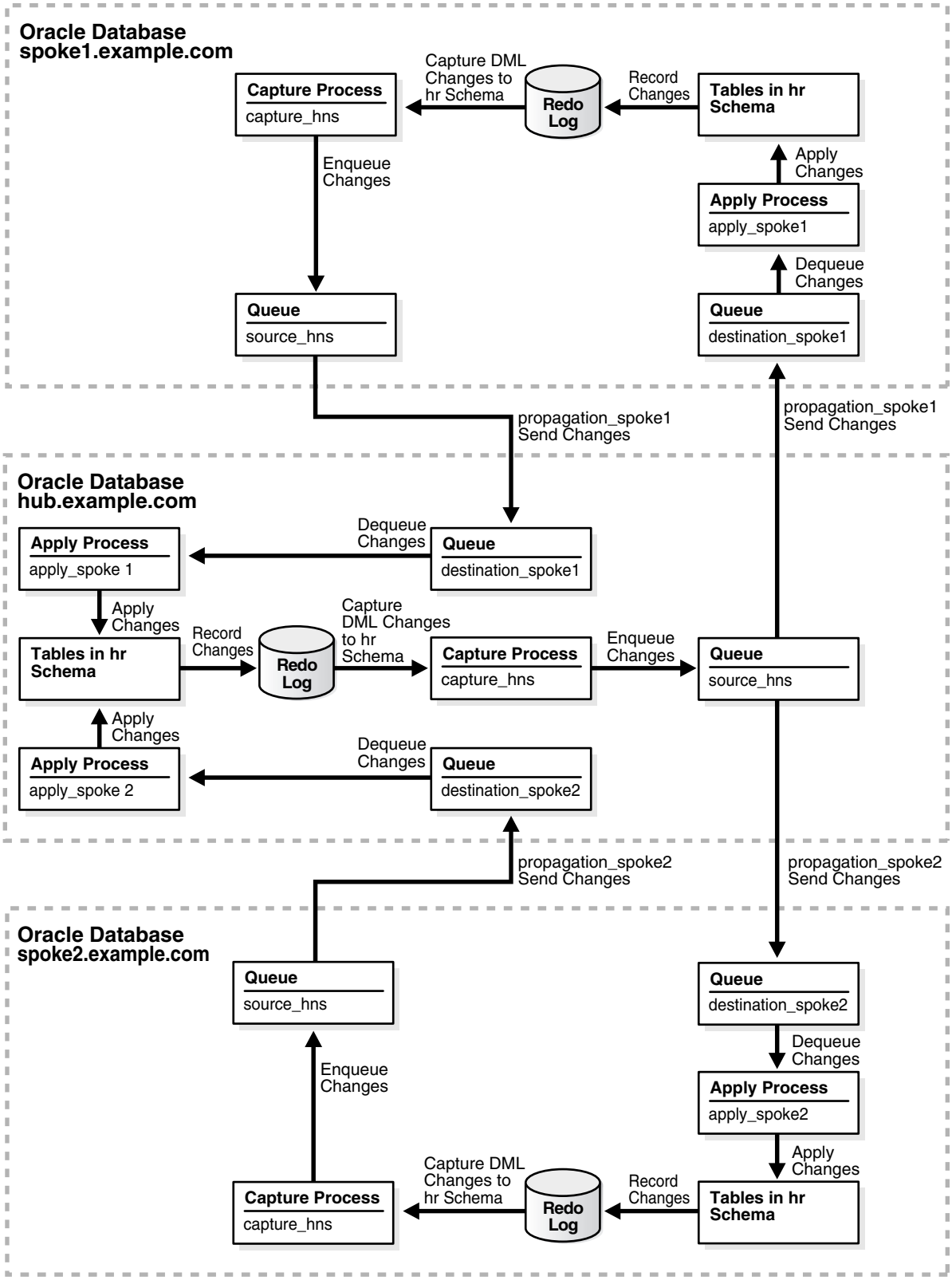
The `MAINTAIN_SCHEMAS` procedure automatically excludes database objects that are not supported by Oracle Streams from the replication environment by adding rules to the negative rule set of each capture and apply process. Query the `DBA_STREAMS_`

UNSUPPORTED data dictionary view to determine which database objects are not supported by Oracle Streams. If unsupported database objects are not excluded, then capture errors will result.

The database objects being configured for replication might or might not exist at the destination databases when you run the MAINTAIN_SCHEMAS procedure. If the database objects do not exist at a destination database, then the MAINTAIN_SCHEMAS procedure instantiates them at the destination database using a Data Pump export/import. During instantiation, the instantiation SCN is set for these database objects. If the database objects already exist at a destination database, then the MAINTAIN_SCHEMAS procedure retains the existing database objects and sets the instantiation SCN for them. In this example, the hr schema exists at each database before the MAINTAIN_SCHEMAS procedure is run.

Figure 2–7 provides an overview of the environment created in this example.

Figure 2–7 Hub-and-Spoke Environment with Capture Processes and Read/Write Spokes



Complete the following steps to use the `MAINTAIN_SCHEMAS` procedure to configure the environment:

1. Complete the following tasks to prepare for the hub-and-spoke replication environment:

   a. Configure network connectivity so that the following databases can communicate with each other:

      – The `hub.example.com` database and the `spoke1.example.com` database

      – The `hub.example.com` database and the `spoke2.example.com` database

      See *Oracle Database 2 Day DBA* for information about configuring network connectivity between databases.

   b. Configure an Oracle Streams administrator at each database that will participate in the replication environment. See "Configuring an Oracle Streams Administrator on All Databases" on page 1-13 for instructions. This example assumes that the Oracle Streams administrator is `strmadmin`.

   c. Create a database link from the hub database to each spoke database and from each spoke database to the hub database. In this example, create the following database links:

      – From the `hub.example.com` database to the `spoke1.example.com` database. Both the name and the service name of the database link must be `spoke1.example.com`.

      – From the `hub.example.com` database to the `spoke2.example.com` database. Both the name and the service name of the database link must be `spoke2.example.com`.

      – From the `spoke1.example.com` database to the `hub.example.com` database. Both the name and the service name of the database link must be `hub.example.com`.

      – From the `spoke2.example.com` database to the `hub.example.com` database. Both the name and the service name of the database link must be `hub.example.com`.

      Each database link should be created in the Oracle Streams administrator's schema. Also, each database link should connect to the Oracle Streams administrator at the destination database. See "Configuring Network Connectivity and Database Links" on page 1-17 for instructions.

   d. Set initialization parameters properly at each database that will participate in the Oracle Streams replication environment. See "Setting Initialization Parameters Relevant to Oracle Streams" on page 1-19 for instructions.

   e. Configure each source database to run in `ARCHIVELOG` mode. For a capture process to capture changes generated at a source database, the source database must be running in `ARCHIVELOG` mode. In this example, all databases must be running in `ARCHIVELOG` mode. See *Oracle Database Administrator's Guide* for information about configuring a database to run in `ARCHIVELOG` mode.

2. Create the following required directory objects:

   - A directory object at the hub database `hub.example.com`. This example assumes that this directory object is `hub_directory`.

- A directory object at the spoke database `spoke1.example.com`. This example assumes that this directory object is `spoke1_directory`.
- A directory object at the spoke database `spoke2.example.com`. This example assumes that this directory object is `spoke2_directory`.

See "Creating the Required Directory Objects" on page 2-15 for instructions.

3. In SQL*Plus, connect to the `hub.example.com` database as the Oracle Streams administrator.

   See *Oracle Database 2 Day DBA* for more information about starting SQL*Plus.

4. Run the `MAINTAIN_SCHEMAS` procedure to configure replication between the `hub.example.com` database and the `spoke1.example.com` database:

```
BEGIN
  DBMS_STREAMS_ADM.MAINTAIN_SCHEMAS(
    schema_names                 => 'hr',
    source_directory_object      => 'hub_directory',
    destination_directory_object => 'spoke1_directory',
    source_database              => 'hub.example.com',
    destination_database         => 'spoke1.example.com',
    capture_name                 => 'capture_hns',
    capture_queue_table          => 'source_hns_qt',
    capture_queue_name           => 'source_hns',
    propagation_name             => 'propagation_spoke1',
    apply_name                   => 'apply_spoke1',
    apply_queue_table            => 'destination_spoke1_qt',
    apply_queue_name             => 'destination_spoke1',
    bi_directional               => TRUE);
END;
/
```

   The `MAINTAIN_SCHEMAS` procedure can take some time to run because it is performing many configuration tasks. Do not allow data manipulation language (DML) or data definition language (DDL) changes to the replicated database objects at the destination database while the procedure is running.

   In the `MAINTAIN_SCHEMAS` procedure, only the following parameters are required: `schema_names`, `source_directory_object`, `destination_directory_object`, `source_database`, and `destination_database`. Also, when you use a configuration procedure to configure bi-directional replication, the `bi_directional` parameter must be set to `TRUE`.

   This example specifies the other parameters to show that you can choose the name for the capture process, capture process's queue table, capture process's queue, propagation, apply process, apply process's queue table, and apply process's queue. If you do not specify these parameters, then system-generated names are used.

   When a configuration procedure is run, information about its progress is recorded in the following data dictionary views: `DBA_RECOVERABLE_SCRIPT`, `DBA_RECOVERABLE_SCRIPT_PARAMS`, `DBA_RECOVERABLE_SCRIPT_BLOCKS`, and `DBA_RECOVERABLE_SCRIPT_ERRORS`. If the procedure stops because it encounters an error, then see *Oracle Streams Replication Administrator's Guide* for instructions about using the `RECOVER_OPERATION` procedure in the `DBMS_STREAMS_ADM` package to recover from these errors.

5. While still connected in SQL*Plus to the `hub.example.com` database as the Oracle Streams administrator, run the `MAINTAIN_SCHEMAS` procedure to

configure replication between the `hub.example.com` database and the `spoke2.example.com` database:

```
BEGIN
  DBMS_STREAMS_ADM.MAINTAIN_SCHEMAS(
    schema_names                 => 'hr',
    source_directory_object      => 'hub_directory',
    destination_directory_object => 'spoke2_directory',
    source_database              => 'hub.example.com',
    destination_database         => 'spoke2.example.com',
    capture_name                 => 'capture_hns',
    capture_queue_table          => 'source_hns_qt',
    capture_queue_name           => 'source_hns',
    propagation_name             => 'propagation_spoke2',
    apply_name                   => 'apply_spoke2',
    apply_queue_table            => 'destination_spoke2_qt',
    apply_queue_name             => 'destination_spoke2',
    bi_directional               => TRUE);
END;
/
```

6. Configure latest time conflict resolution for all of the tables in the `hr` schema at the `hub.example.com`, `spoke1.example.com`, and `spoke2.example.com` databases. This schema includes the `countries`, `departments`, `employees`, `jobs`, `job_history`, `locations`, and `regions` tables. See Chapter 9, "Oracle Streams Conflict Resolution" for instructions.

   See Also: *Oracle Database 2 Day + Data Replication and Integration Guide* for an example that configures this replication environment using Oracle Enterprise Manager

Monitoring Oracle Streams Configuration Progress

The following procedures in the `DBMS_STREAMS_ADM` package configure a replication environment that is maintained by Oracle Streams:

- `MAINTAIN_GLOBAL`
- `MAINTAIN_SCHEMAS`
- `MAINTAIN_SIMPLE_TTS`
- `MAINTAIN_TABLES`
- `MAINTAIN_TTS`
- `PRE_INSTANTIATION_SETUP` and `POST_INSTANTIATION_SETUP`

While one of these procedures configures the replication environment directly (with the `perform_actions` parameter is set to `TRUE`), you can monitor the progress of the configuration in a separate terminal window.

Complete the following steps to monitor the progress of the Oracle Stream configuration:

1. In SQL*Plus, connect to the capture database as the Oracle Streams administrator. Use a different terminal window than the one that is running the configuration procedure.

   See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.

2. For basic information about the configuration operation, run the following query:

```
COLUMN SCRIPT_ID      HEADING 'Script ID'      FORMAT A40
COLUMN CREATION_TIME  HEADING 'Creation|Time'  FORMAT A20

SELECT SCRIPT_ID,
  TO_CHAR(CREATION_TIME,'HH24:Mi:SS MM/DD/YY') CREATION_TIME
  FROM DBA_RECOVERABLE_SCRIPT;
```

Your output is similar to the following:

```
                                        Creation
Script ID                               Time
---------------------------------------- --------------------
64EE0DFCC374CE7EE040578C89174D3E         07:46:54 03/12/09
```

This output shows the script ID for the configuration operation and the time when the operation started.

3. For detailed information about the progress of the configuration operation, run the following query:

```
COLUMN STATUS           HEADING 'Status'          FORMAT A12
COLUMN PROGRESS         HEADING 'Steps|Completed' FORMAT A10
COLUMN ELAPSED_SECONDS  HEADING 'Elapsed|Seconds' FORMAT A10
COLUMN CURRENT_STEP     HEADING 'Current Step'    FORMAT A20
COLUMN PROCEDURE        HEADING 'Procedure'       FORMAT A20

SELECT  rs.STATUS,
    rs.DONE_BLOCK_NUM||' of '||rs.TOTAL_BLOCKS PROGRESS,
    TO_CHAR(TO_NUMBER(SYSDATE-rs.CREATION_TIME)*86400,9999.99) ELAPSED_SECONDS,
    SUBSTR(TO_CHAR(rsb.FORWARD_BLOCK),1,100) CURRENT_STEP,
    rs.INVOKING_PACKAGE||'.'||rs.INVOKING_PROCEDURE PROCEDURE
  FROM DBA_RECOVERABLE_SCRIPT rs, DBA_RECOVERABLE_SCRIPT_BLOCKS rsb
  WHERE rs.SCRIPT_ID  = rsb.SCRIPT_ID AND
        rsb.BLOCK_NUM = rs.DONE_BLOCK_NUM + 1;
```

Your output is similar to the following:

```
             Steps      Elapsed
Status       Completed  Seconds    Current Step         Procedure
------------ ---------- ---------- -------------------- --------------------
EXECUTING    7 of 39    132        --                   DBMS_STREAMS_ADM.MAI
                                   -- Set up queue "STR NTAIN_SCHEMAS
                                   MADMIN"."PROD$APPQ"
                                   --
                                   BEGIN
                                     dbms_streams_adm.s
                                   et_up_queue(
                                       queue_ta
```

This output shows the following information about the configuration operation:

- The current status of the configuration operation, either `GENERATING`, `NOT EXECUTED`, `EXECUTING`, `EXECUTED`, or `ERROR`
- The number of steps completed and the total number of steps required to complete the operation
- The amount of time, in seconds, that the configuration operation has been running
- The operation being performed by the current step
- The PL/SQL procedure being executed in the current step

See Also: "Recovering from Operation Errors" on page 12-36

3

Flexible Oracle Streams Replication Configuration

This chapter describes flexible methods for configuring Oracle Streams replication between two or more databases. This chapter includes step-by-step instructions for configuring each Oracle Streams component to build a single-source or multiple-source replication environment.

One common type of single-source replication environment is a hub-and-spoke replication environment that does not allow changes to the replicated database objects in the spoke databases. The following are common types of multiple-source replication environments:

- A hub-and-spoke replication environment that allows changes to the replicated database objects in the spoke databases
- An n-way replication environment

Oracle Database 2 Day + Data Replication and Integration Guide describes these common types of replication environments in detail.

If possible, consider using a simple method for configuring Oracle Streams replication described in Chapter 2, "Simple Oracle Streams Replication Configuration". You can either use the Oracle Streams tool in Enterprise Manager or a single procedure in the `DBMS_STREAMS_ADM` package configure all of the Oracle Streams components in a replication environment with two databases. Also, you can use a simple method and still meet custom requirements for your replication environment in one of the following ways:

- You can use a simple method to generate a configuration script and modify the script to meet your requirements.
- You can use a simple method to configure Oracle Streams replication between two databases and add new database objects or databases to the environment by following the instructions in Chapter 4, "Adding to an Oracle Streams Replication Environment".

However, if you require more flexibility in your Oracle Streams replication configuration than what is available with the simple methods, then you can follow the instructions in this chapter to configure the environment.

This chapter contains these topics:

- Creating a New Oracle Streams Single-Source Environment
- Creating a New Oracle Streams Multiple-Source Environment

Note:

- The instructions in the following sections assume you will use the `DBMS_STREAMS_ADM` package to configure your Oracle Streams environment. If you use other packages, then extra steps might be necessary for each task.
- Certain types of database objects are not supported by Oracle Streams. When you configure an Oracle Streams environment, ensure that no capture process attempts to capture changes to an unsupported database object. Also, ensure that no synchronous capture or apply process attempts to process changes to unsupported columns. To list unsupported database objects and unsupported columns, query the `DBA_STREAMS_UNSUPPORTED` and `DBA_STREAMS_COLUMNS` data dictionary views.

See Also: *Oracle Streams Concepts and Administration* for instructions on determining which database objects are not supported by Oracle Streams

Creating a New Oracle Streams Single-Source Environment

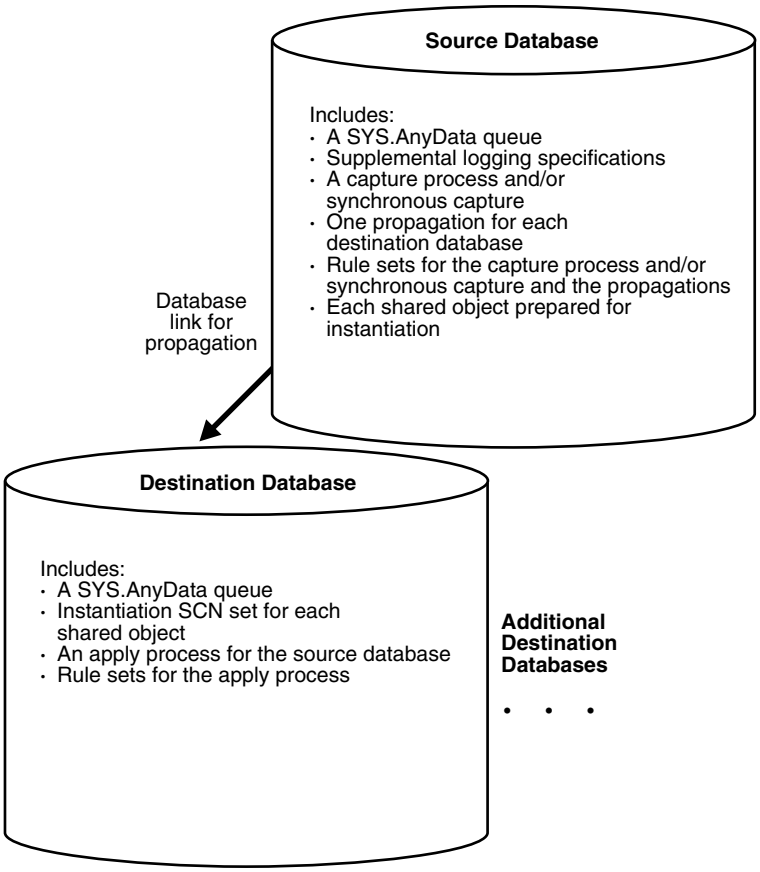
This section lists the general steps to perform when creating a new single-source Oracle Streams environment. A single-source environment is one in which there is only one source database for replicated data. There can be multiple source databases in a single-source environment, but no two source databases capture any of the same data. A one-way replication environment with two databases is an example of a single-source environment.

Before starting capture processes, creating synchronous captures, and configuring propagations in a new Oracle Streams environment, ensure that any propagations or apply processes that will receive LCRs are configured to handle these LCRs. That is, the propagations or apply processes should exist, and each one should be associated with rule sets that handle the LCRs appropriately. If these propagations and apply processes are not configured properly to handle these LCRs, then LCRs can be lost.

This example assumes that the replicated database objects are read-only at the destination databases. If the replicated database objects are read/write at the destination databases, then the replication environment will not stay synchronized because Oracle Streams is not configured to replicate the changes made to the replicated objects at the destination databases.

Figure 3–1 shows an example Oracle Streams single-source replication environment.

Figure 3–1 Example Oracle Streams Single-Source Environment



You can create an Oracle Streams replication environment that is more complicated than the one shown in Figure 3–1. For example, a single-source Oracle Streams replication environment can use downstream capture and directed networks.

In general, if you are configuring a new Oracle Streams single-source environment in which changes to replicated database 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. Make the necessary decisions about configuring the replication environment. See "Decisions to Make Before Configuring Oracle Streams Replication" on page 1-4.
2. Complete the necessary tasks to prepare each database in your environment for Oracle Streams. See "Tasks to Complete Before Configuring Oracle Streams Replication" on page 1-13.

   Some of these tasks might not be required at certain databases.
3. Create any necessary `ANYDATA` queues that do not already exist. When you create a capture process, synchronous capture, or apply process, you associate the process with a specific `ANYDATA` queue. When you create a propagation, you associate it with a specific source queue and destination queue. See "Creating an ANYDATA Queue" on page 6-1 for instructions.
4. Specify supplemental logging at each source database for any replicated database object. See "Specifying Supplemental Logging" on page 1-28 for instructions.
5. At each database, create the required capture processes, synchronous captures, propagations, and apply processes for your environment. You can create capture

processes, propagations, and apply processes in any order. If you create synchronous captures, then create them after you create the relevant propagations and apply processes.

- Create one or more capture processes at each database that will capture changes with a capture process. Ensure that each capture process uses rule sets that are appropriate for capturing changes. Do not start the capture processes you create. Oracle recommends that you use only one capture process for each source database. See "Configuring a Capture Process" on page 5-1 for instructions.

  When you use a procedure in the `DBMS_STREAMS_ADM` package to add the capture process 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, if the capture process is a local capture process or a downstream capture process with a database link to the source database.

  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 rules.

  – You use an existing capture process and do not add capture process rules for any replicated object.

  – You use a downstream capture process with no database link to the source database.

  If you must prepare for instantiation manually, then see "Preparing Database Objects for Instantiation at a Source Database" on page 8-8 for instructions.

- Create all propagations that send the captured LCRs from a source queue to a destination queue. Ensure that each propagation uses rule sets that are appropriate for sending changes. See "Creating Oracle Streams Propagations Between ANYDATA Queues" on page 6-3 for instructions.

- Create one or more apply processes at each database that will apply changes. Ensure that each apply process uses rule sets that are appropriate for applying changes. Do not start the apply processes you create. See Chapter 7, "Configuring Implicit Apply" for instructions.

- Create one or more synchronous captures at each database that will capture changes with a synchronous capture. Ensure that each synchronous capture use a rule set that is appropriate for capturing changes. Do not create the synchronous capture until you create all of the propagations and apply processes that will process its LCRs. See "Configuring Synchronous Capture" on page 5-17 for instructions.

  When you use a procedure in the `DBMS_STREAMS_ADM` package to add the synchronous capture rules, it automatically runs the `PREPARE_SYNC_INSTANTIATION` function in the `DBMS_CAPTURE_ADM` package for the specified table.

6. Either instantiate, or set the instantiation SCN for, each database object for which changes are applied by an apply process. If the database objects do not exist at a destination database, then instantiate them using export/import, transportable tablespaces, or RMAN. If the database objects already exist at a destination database, then set the instantiation SCNs for them manually.

- To instantiate database objects using export/import, first export them at the source database. Next, import them at the destination database. See Chapter 8, "Instantiation and Oracle Streams Replication".

  Do not allow any changes to the database objects being exported during export at the source database. Do not allow changes to the database objects being imported during import at the destination database.

  You can specify a more stringent degree of consistency by using an export parameter such as `FLASHBACK_SCN` or `FLASHBACK_TIME`.

- 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 replicated 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 set the `recursive` parameter for this procedure to `TRUE` so that the instantiation SCN also is set for each schema at the destination database and for the tables owned by these schemas.

  If you run `SET_SCHEMA_INSTANTIATION_SCN` at a destination database, then set the `recursive` parameter for this procedure to `TRUE` so that the instantiation SCN also is set for each table in the schema.

  If you set the `recursive` parameter to `TRUE` in the `SET_GLOBAL_INSTANTIATION_SCN` procedure or the `SET_SCHEMA_INSTANTIATION_SCN` procedure, then a database link from the destination database to the source database is required. This database link must have the same name as the global name of the source database and must be accessible to the user who executes the procedure. See "Setting Instantiation SCNs Using the DBMS_APPLY_ADM Package" on page 8-38 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 ensure that no rows are imported. Also, ensure that the replicated database 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 8-37 for more information about performing a metadata export/import.

7. Start each apply process you created in Step 5 using the `START_APPLY` procedure in the `DBMS_APPLY_ADM` package.

8. Start each capture process you created in Step 5 using the `START_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package.

When you are configuring the environment, remember that capture processes and apply processes are stopped when they are created. However, synchronous captures start to capture changes immediately when they are created, and propagations are scheduled to send LCRs immediately when they are created. A capture process or

synchronous capture 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 or creating the synchronous capture, and you must instantiate the objects before running the whole stream.

See Also:

- *Oracle Streams Extended Examples* for detailed examples that set up single-source environments

Creating a New Oracle Streams Multiple-Source Environment

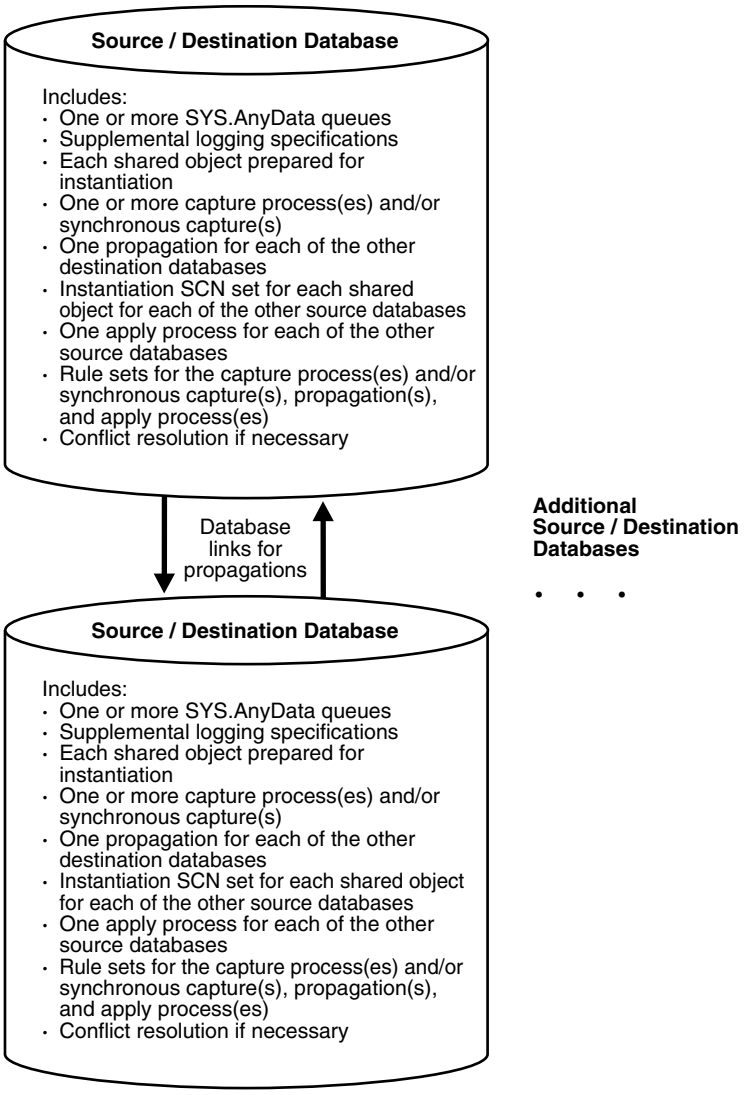
This section lists the general steps to perform when creating a new multiple-source Oracle Streams environment. A multiple-source environment is one in which there are multiple source databases for any of the replicated data. An n-way replication environment is an example of a multiple-source environment.

This example uses the following terms:

- **Populated database:** A database that already contains the replicated database objects before you create the new multiple-source environment. You must have at least one populated database to create the new Oracle Streams environment.
- **Export database:** A populated database on which you perform an export of the replicated database objects. This export is used to instantiate the replicated database objects at the import databases. You might 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 replicated database objects before you create the new multiple-source environment. You instantiate the replicated database objects at an import database by performing an import of these database objects. You might not have any import databases if all of the databases in the environment are populated databases.

Figure 3–2 shows an example multiple-source Oracle Streams environment.

Figure 3–2 Example Oracle Streams Multiple-Source Environment



You can create an Oracle Streams replication environment that is more complicated than the one shown in Figure 3–2. For example, a multiple-source Oracle Streams replication environment can use downstream capture and directed networks.

When there are multiple source databases in an Oracle Streams replication environment, change cycling is possible. Change cycling happens when a change is sent back to the database where it originated. Typically, you should avoid change cycling. Before you configure your replication environment, see Chapter 10, "Oracle Streams Tags", and ensure that you configure the replication environment to avoid change cycling.

Complete the following steps to create a multiple-source environment:

Note: Ensure that no changes are made to the objects being shared at a database you are adding to the Oracle Streams environment until the instantiation at the database is complete.

---

1. Make the necessary decisions about configuring the replication environment. See "Decisions to Make Before Configuring Oracle Streams Replication" on page 1-4.
2. Complete the necessary tasks to prepare each database in your environment for Oracle Streams. See "Tasks to Complete Before Configuring Oracle Streams Replication" on page 1-13.

   Some of these tasks might not be required at certain databases.
3. At each populated database, specify any necessary supplemental logging for the replicated database objects. See "Specifying Supplemental Logging" on page 1-28 for instructions.
4. Create any necessary `ANYDATA` queues that do not already exist. When you create a capture process, synchronous capture, or apply process, you associate the process with a specific `ANYDATA` queue. When you create a propagation, you associate it with a specific source queue and destination queue. See "Creating an ANYDATA Queue" on page 6-1 for instructions.
5. At each database, create the required capture processes, synchronous captures, propagations, and apply processes for your environment. You can create capture processes, propagations, and apply processes in any order. If you create synchronous captures, then create them after you create the relevant propagations and apply processes.

   - Create one or more capture processes at each database that will capture changes with a capture process. Ensure that each capture process uses rule sets that are appropriate for capturing changes. Do not start the capture processes you create. Oracle recommends that you use only one capture process for each source database. See "Configuring a Capture Process" on page 5-1 for instructions.

     When you use a procedure in the `DBMS_STREAMS_ADM` package to add the capture process 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, if the capture process is a local capture process or a downstream capture process with a database link to the source database.

     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 rules.

     – You use an existing capture process and do not add capture process rules for any replicated database object.

     – You use a downstream capture process with no database link to the source database.

     If you must prepare for instantiation manually, then see "Preparing Database Objects for Instantiation at a Source Database" on page 8-8 for instructions.

   - Create all propagations that propagate the captured LCRs from a source queue to a destination queue. Ensure that each propagation uses rule sets that are appropriate for propagating changes. See "Creating Oracle Streams Propagations Between ANYDATA Queues" on page 6-3 for instructions.

   - Create one or more apply processes at each database that will apply changes. Ensure that each apply process uses rule sets that are appropriate for applying

changes. Do not start the apply processes you create. See Chapter 7, "Configuring Implicit Apply" for instructions.

- Create one or more synchronous captures at each database that will capture changes with a synchronous capture. Ensure that each synchronous capture uses rule sets that are appropriate for capturing changes. Do not create the synchronous capture until you create all of the propagations and apply processes that will process its LCRs. See "Configuring Synchronous Capture" on page 5-17 for instructions.

  When you use a procedure in the `DBMS_STREAMS_ADM` package to add the synchronous capture rules, it automatically runs the `PREPARE_SYNC_INSTANTIATION` function in the `DBMS_CAPTURE_ADM` package for the specified table.

After completing these steps, complete the steps in each of the following sections that apply to your environment. You might 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 3-9. These steps are required only if your environment has multiple populated databases.
- For each import database, complete the steps in "Adding Replicated Objects to Import Databases When Creating a New Environment" on page 3-10.

Configuring Populated Databases When Creating a Multiple-Source Environment

After completing the steps in "Creating a New Oracle Streams Multiple-Source Environment" on page 3-6, complete the following steps for the populated databases if your environment has multiple populated databases:

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:

   - Perform a metadata only export of the replicated database 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. Ensure that no rows are imported. Also, ensure that the replicated database 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 replicating DML changes only, then table level export/import is sufficient. If you are replicating DDL changes also, then additional considerations apply. See "Setting Instantiation SCNs Using Export/Import" on page 8-37 for more information about performing a metadata export/import.

   - Set the instantiation SCNs manually at each of the other populated databases. Do this for each of the replicated database objects. Ensure that the replicated database 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 8-38 for instructions.

Adding Replicated Objects to Import Databases When Creating a New Environment

After completing the steps in "Creating a New Oracle Streams Multiple-Source Environment" on page 3-6, 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. In this case, the import database will be the source database for these destination databases. The databases where you set the instantiation SCNs can 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 replicated 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 replicated 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 Using the DBMS_APPLY_ADM Package" on page 8-38 for instructions.

   Because you run these procedures before any tables are instantiated at the import databases, and because the local capture processes or synchronous captures are configured already for these import databases, you will not need to run the `SET_TABLE_INSTANTIATION_SCN` procedure for each table created during the instantiation. Instantiation SCNs will be set automatically for these tables at all of the other databases in the environment that will be destination databases of the import database.
3. At the export database you chose in Step 1, perform an export of the replicated database objects. Next, perform an import of the replicated database objects at each import database. See Chapter 8, "Instantiation and Oracle Streams Replication" and *Oracle Database Utilities* for information about using export/import.

   Do not allow any changes to the database objects being exported while exporting these database objects at the source database. Do not allow changes to the database objects being imported while importing these database objects at the destination database.

   You can specify a more stringent degree of consistency by using an export parameter such as `FLASHBACK_SCN` or `FLASHBACK_TIME`.
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:

- 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 replicated database 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 8-37 for more information about performing a metadata export/import.

- For each populated database, set the instantiation SCN manually for each replicated database object at each import database. Ensure that the replicated database 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 8-38 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 Oracle Streams Multiple-Source Environment" on page 3-6
- "Configuring Populated Databases When Creating a Multiple-Source Environment" on page 3-9, if your environment has multiple populated databases
- "Adding Replicated Objects to Import Databases When Creating a New Environment" on page 3-10, if your environment has one or more import databases

When all of the previous configuration steps are finished, complete the following steps:

1. At each database, configure conflict resolution if conflicts are possible. See Chapter 9, "Oracle Streams Conflict Resolution" for instructions.
2. Start each apply process in the environment using the `START_APPLY` procedure in the `DBMS_APPLY_ADM` package.
3. Start each capture process the environment using the `START_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package.

See Also: *Oracle Streams Extended Examples* for a detailed example that creates a multiple-source environment

4

Adding to an Oracle Streams Replication Environment

This chapter contains instructions for adding database objects and databases to an existing Oracle Streams replication environment.

This chapter contains these topics:

- About Adding to an Oracle Streams Replication Environment
- Adding Multiple Components Using a Single Procedure
- Adding Components Individually in Multiple Steps

Note: Certain types of database objects are not supported by Oracle Streams. When you extend an Oracle Streams environment, ensure that no capture process attempts to capture changes to an unsupported database object. Also, ensure that no synchronous capture or apply process attempts to process changes to unsupported columns. To list unsupported database objects and unsupported columns, query the `DBA_STREAMS_UNSUPPORTED` and `DBA_STREAMS_COLUMNS` data dictionary views.

See Also:

- Chapter 2, "Simple Oracle Streams Replication Configuration"
- Chapter 3, "Flexible Oracle Streams Replication Configuration"
- *Oracle Streams Concepts and Administration* for instructions on determining which database objects are not supported by Oracle Streams

About Adding to an Oracle Streams Replication Environment

Sometimes it is necessary to extend an Oracle Streams replication environment when the needs of your organization change. You can extend an Oracle Streams replication environment by adding database objects or databases.

There are three ways to extend an Oracle Streams replication environment:

- About Using the Setup Streams Replication Wizard or a Single Configuration Procedure
- About Adding the Oracle Streams Components Individually in Multiple Steps

About Using the Setup Streams Replication Wizard or a Single Configuration Procedure

There are two easy ways to extend an Oracle Streams replication environment:

- Run the Setup Streams Replication Wizard in Oracle Enterprise Manager
- Run one of the following procedures in the `DBMS_STREAMS_ADM` package:
  - The `MAINTAIN_GLOBAL` procedure can add a new database to an environment that replicates changes to all of the database objects in the databases.
  - The `MAINTAIN_SCHEMAS` procedure can add one or more new schemas to the existing databases in the replication environment, or it can add a new database that replicates schemas that are currently being replicated.
  - The `MAINTAIN_SIMPLE_TTS` procedure can add a new simple tablespace to an existing replication environment, or it can add a new database that replicates a simple tablespace that is currently being replicated.
  - The `MAINTAIN_TABLES` procedure can add one or more new tables to the existing databases in the replication environment, or it can add a new database that replicates tables that are currently being replicated.
  - The `MAINTAIN_TTS` procedure can add a new set of tablespaces to an existing replication environment, or it can add a new database that replicates a set of tablespaces that are currently being replicated.

To use either of these methods to extend an Oracle Streams replication environment, the environment must meet the following conditions:

- It must be a two-database or hub-and-spoke replication environment that was configured by the Setup Streams Replication Wizard or by one of the configuration procedures in the `DBMS_STREAMS_ADM` package. See *Oracle Database 2 Day + Data Replication and Integration Guide* for information about these types of replication environments.
- It cannot use a synchronous capture at any database in the Oracle Streams replication environment. See *Oracle Streams Concepts and Administration* for more information about synchronous capture.
- If you are adding a database to the environment, then each database that captures changes must use a local capture process. No database can use a downstream capture process. If you are adding one or more database objects to the environment, then the databases can use either local or downstream capture processes. See "Decide Whether to Configure Local or Downstream Capture for the Source Database" on page 1-5 for more information about downstream capture.
- If you are adding database objects to the replication environment, then the database objects must exist at the database specified in the `source_database` parameter of the configuration procedure.

If your environment meets these conditions, then you can use the Setup Streams Replication Wizard or a single procedure to extend the environment.

The following are additional requirements for cases in which the replicated database objects already exist at an intended destination database before you run the wizard or procedure:

- If you are adding database objects to the replication environment, and one or more of these database objects exist at a database other than the source database, then meet the following requirements:

  - Before running the wizard or procedure, ensure that the replicated database objects at each destination database are consistent with replicated database objects at the source database.
  - After running the wizard or procedure, ensure that the instantiation SCN is set for each replicated database object at each destination database. See "Setting Instantiation SCNs at a Destination Database" on page 8-36 and "Monitoring Instantiation""Monitoring Instantiation".
- If you are adding a database to the replication environment, then any of the database objects that are replicated in the current environment exist at the added database, then meet the following requirements:
  - Before running the wizard or procedure, ensure that the replicated database objects at each database being added are consistent with replicated database objects at the source database.
  - After running the wizard or procedure, ensure that the instantiation SCN is set for each replicated database object at the added database. See "Setting Instantiation SCNs at a Destination Database" on page 8-36 and "Monitoring Instantiation""Monitoring Instantiation".

For instructions about adding to a replication environment using the wizard or a single procedure, see the following documentation:

- *Oracle Database 2 Day + Data Replication and Integration Guide* for instructions about using the Setup Streams Replication Wizard
- "Adding Multiple Components Using a Single Procedure" on page 4-3 for instructions about using a single procedure in the `DBMS_STREAMS_ADM` package

See Also: *Oracle Database PL/SQL Packages and Types Reference* for information about the procedures in the `DBMS_STREAMS_ADM` chapter

About Adding the Oracle Streams Components Individually in Multiple Steps

If you cannot extend the Oracle Streams replication environment by using the Setup Streams Replication Wizard or a configuration procedure in the `DBMS_STREAMS_ADM` package, then you must complete the configuration steps manually. These steps include adding the necessary rules and Oracle Streams components to the environment, and other configuration steps.

If you must extend the Oracle Streams replication environment manually, then see the instructions in "Adding Components Individually in Multiple Steps" on page 4-10.

Adding Multiple Components Using a Single Procedure

This section describes adding Oracle Streams components a single PL/SQL procedure in the `DBMS_STREAMS_ADM` package. Oracle Streams components include queues, rules, rule sets, capture processes, synchronous captures, propagations, and apply processes.

This section contains these topics:

- Adding Database Objects to a Replication Environment Using a Single Procedure
- Adding a Database to a Replication Environment Using a Single Procedure

Adding Database Objects to a Replication Environment Using a Single Procedure

This topic includes an example that uses the `MAINTAIN_TABLES` procedure in the `DBMS_STREAMS_ADM` package to add tables to an existing hub-and-spoke replication environment. When the example is complete, the Oracle Streams replication environment replicates the changes made to the added tables at the databases in the environment.

Specifically, the example in this topic extends the replication environment configured in "Example That Configures Hub-and-Spoke Replication" on page 2-48. That configuration has the following characteristics:

- The `hr` schema is replicated at the `hub.example.com`, `spoke1.example.com`, and `spoke2.example.com` databases.
- The `hub.example.com` database is the hub database in the hub-and-spoke environment, while the other databases are the spoke databases.
- The spoke databases allow changes to the replicated schema, and each database has a local capture process to capture these changes.
- Update conflict handlers are configured for each replicated table at each database to resolve conflicts

This example adds the following tables to the environment:

- `oe.orders`
- `oe.order_items`

This example uses the tables in the `oe` sample schema. The `oe` sample schema is installed by default with Oracle Database.

> **Note:** Before you use a configuration procedure in the `DBMS_STREAMS_ADM` package to extend an Oracle Streams replication environment, ensure that the environment meets the conditions described in "About Using the Setup Streams Replication Wizard or a Single Configuration Procedure" on page 4-2.

Complete the following steps:

1. Ensure that the following directory objects exist, and remove any files related to the previous configuration from them, including Data Pump export dump files and export log files:
   - The `hub_dir` directory object at the `hub.example.com` database.
   - The `spoke1_dir` directory object at the `spoke1.example.com` database.
   - The `spoke2_dir` directory object at the `spoke2.example.com` database.
2. Stop the capture process at the hub database in the hub-and-spoke environment.

   Use the `STOP_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package to stop a capture process, or see *Oracle Database 2 Day + Data Replication and Integration Guide* for instructions about stopping a capture process using Oracle Enterprise Manager.

   In this example, stop the capture process at the `hub.example.com` database. The replicated database objects can remain open to changes while the capture process is stopped. These changes will be captured when the capture process is restarted.

3. In SQL*Plus, run the appropriate configuration procedure in the DBMS_STREAMS_ADM package at the hub database to add each new database object for each spoke database.

   You might need to run the procedure several times if the environment has multiple spoke databases. In this example, complete the following steps:

   a. Open SQL*Plus and connect to the hub.example.com database as the Oracle Streams administrator.

      See *Oracle Database 2 Day DBA* for more information about starting SQL*Plus.

   b. Run the MAINTAIN_TABLES procedure to add the oe.orders and oe.order_items tables for replication between hub.example.com and spoke1.example.com:

```
DECLARE
  tables DBMS_UTILITY.UNCL_ARRAY;
  BEGIN
    tables(1) := 'oe.orders';
    tables(2) := 'oe.order_items';
    DBMS_STREAMS_ADM.MAINTAIN_TABLES(
      table_names                 => tables,
      source_directory_object     => 'hub_dir',
      destination_directory_object => 'spoke1_dir',
      source_database             => 'hub.example.com',
      destination_database        => 'spoke1.example.com',
      capture_name                => 'capture_hns',
      capture_queue_table         => 'source_hns_qt',
      capture_queue_name          => 'source_hns',
      propagation_name            => 'propagation_spoke1',
      apply_name                  => 'apply_spoke1',
      apply_queue_table           => 'destination_spoke1_qt',
      apply_queue_name            => 'destination_spoke1',
      bi_directional              => TRUE);
END;
/
```

      The MAINTAIN_TABLES procedure can take some time to run because it is performing many configuration tasks. Do not allow data manipulation language (DML) or data definition language (DDL) changes to the specified tables at the destination database while the procedure is running. When the procedure completes, the new database objects are added to the environment, and the capture process that was stopped in Step 2 is restarted.

      When a configuration procedure is run, information about its progress is recorded in the following data dictionary views: DBA_RECOVERABLE_SCRIPT, DBA_RECOVERABLE_SCRIPT_PARAMS, DBA_RECOVERABLE_SCRIPT_BLOCKS, and DBA_RECOVERABLE_SCRIPT_ERRORS. If the procedure stops because it encounters an error, then see *Oracle Streams Replication Administrator's Guide* for instructions about using the RECOVER_OPERATION procedure in the DBMS_STREAMS_ADM package to recover from these errors.

      The parameter values that specify Oracle Streams component names must be the same as the values specified in the configuration procedure in the DBMS_STREAMS_ADM package that configured the replication environment. The Oracle Streams component names specified include the capture process name, queue names, queue table names, the propagation name, and the apply process name. In this example, the Oracle Streams component names match

the ones specified in "Example That Configures Hub-and-Spoke Replication" on page 2-48.

c. Run the `MAINTAIN_TABLES` procedure to add the `oe.orders` and `oe.order_items` tables for replication between `hub.example.com` and `spoke2.example.com`:

```
DECLARE
  tables DBMS_UTILITY.UNCL_ARRAY;
  BEGIN
    tables(1) := 'oe.orders';
    tables(2) := 'oe.order_items';
    DBMS_STREAMS_ADM.MAINTAIN_TABLES(
      table_names                 => tables,
      source_directory_object     => 'hub_dir',
      destination_directory_object => 'spoke2_dir',
      source_database             => 'hub.example.com',
      destination_database        => 'spoke2.example.com',
      capture_name                => 'capture_hns',
      capture_queue_table         => 'source_hns_qt',
      capture_queue_name          => 'source_hns',
      propagation_name            => 'propagation_spoke2',
      apply_name                  => 'apply_spoke2',
      apply_queue_table           => 'destination_spoke2_qt',
      apply_queue_name            => 'destination_spoke2',
      bi_directional              => TRUE);
END;
/
```

4. Set the instantiation SCN for the replicated tables at the spoke databases:

Note: This step is required in this example because the replicated tables existed at the spoke databases before the `MAINTAIN_TABLES` procedure was run. If the replicated tables did not exist at the spoke databases before the `MAINTAIN_TABLES` procedure was run, then the procedure sets the instantiation SCN for the replicated tables and this step is not required. Ensure that the data in the shared table is consistent at the source and destination databases when the instantiation SCN is set and that no changes are made to the table at the source database until after the SCN that is used for the instantiation SCN.

---

a. In SQL*Plus, connect to the `hub.example.com` database as the Oracle Streams administrator.

See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

b. Set the instantiation SCN for the `oe.orders` table at the `spoke1.example.com` database:

```
DECLARE
  iscn  NUMBER;    -- Variable to hold instantiation SCN value
BEGIN
  iscn := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER();
  DBMS_APPLY_ADM.SET_TABLE_INSTANTIATION_SCN@spoke1.example.com(
    source_object_name    => 'oe.orders',
    source_database_name  => 'hub.example.com',
    instantiation_scn     => iscn);
```

```
END;
/
```

c. Set the instantiation SCN for the `oe.order_items` table at the `spoke1.example.com` database:

```
DECLARE
  iscn  NUMBER;    -- Variable to hold instantiation SCN value
BEGIN
  iscn := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER();
  DBMS_APPLY_ADM.SET_TABLE_INSTANTIATION_SCN@spoke1.example.com(
    source_object_name    => 'oe.order_items',
    source_database_name  => 'hub.example.com',
    instantiation_scn     => iscn);
END;
/
```

d. Set the instantiation SCN for the `oe.orders` table at the `spoke2.example.com` database:

```
DECLARE
  iscn  NUMBER;    -- Variable to hold instantiation SCN value
BEGIN
  iscn := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER();
  DBMS_APPLY_ADM.SET_TABLE_INSTANTIATION_SCN@spoke2.example.com(
    source_object_name    => 'oe.orders',
    source_database_name  => 'hub.example.com',
    instantiation_scn     => iscn);
END;
/
```

e. Set the instantiation SCN for the `oe.order_items` table at the `spoke2.example.com` database:

```
DECLARE
  iscn  NUMBER;    -- Variable to hold instantiation SCN value
BEGIN
  iscn := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER();
  DBMS_APPLY_ADM.SET_TABLE_INSTANTIATION_SCN@spoke2.example.com(
    source_object_name    => 'oe.order_items',
    source_database_name  => 'hub.example.com',
    instantiation_scn     => iscn);
END;
/
```

5. Configure latest time conflict resolution for the `orders` and `order_items` tables in the `oe` schema at the `hub.example.com`, `spoke1.example.com`, and `spoke2.example.com` databases. See *Oracle Database 2 Day + Data Replication and Integration Guide* for instructions.

Adding a Database to a Replication Environment Using a Single Procedure

This topic includes an example that uses the `MAINTAIN_SCHEMAS` procedure in the `DBMS_STREAMS_ADM` package to add a new spoke database to an existing hub-and-spoke replication environment. When the example is complete, the Oracle Streams replication environment replicates the changes made to the schema with the new database.

Specifically, the example in this topic extends the replication environment configured in "Example That Configures Hub-and-Spoke Replication" on page 2-48. That configuration has the following characteristics:

- The `hr` schema is replicated at the `hub.example.com`, `spoke1.example.com`, and `spoke2.example.com` databases.
- The `hub.example.com` database is the hub database in the hub-and-spoke environment, while the other databases are the spoke databases.
- The spoke databases allow changes to the replicated schema, and each database has a local capture process to capture these changes.

This example adds the `spoke3.example.com` database to the environment.

Note: Before you use a configuration procedure in the `DBMS_STREAMS_ADM` package to extend an Oracle Streams replication environment, ensure that the environment meets the conditions described in "About Using the Setup Streams Replication Wizard or a Single Configuration Procedure" on page 4-2.

Complete the following steps:

1. Complete the following tasks to prepare the environment for the new database:

   a. Configure network connectivity so that the hub database can communicate with the new spoke database. In this example, configure network connectivity so that the `hub.example.com` database and the `spoke3.example.com` databases can communicate with each other.

      See *Oracle Database 2 Day DBA* for information about configuring network connectivity between databases.

   b. Configure an Oracle Streams administrator at the new spoke database. In this example, configure an Oracle Streams administrator at the `spoke3.example.com` database. See "Configuring an Oracle Streams Administrator on All Databases" on page 1-13 for instructions. This example assumes that the Oracle Streams administrator is `strmadmin`.

   c. Create a database link from the hub database to new spoke database and from new spoke database to the hub database. In this example, create the following database links:

      – From the `hub.example.com` database to the `spoke3.example.com` database. Both the name and the service name of the database link must be `spoke3.example.com`.

      – From the `spoke3.example.com` database to the `hub.example.com` database. Both the name and the service name of the database link must be `hub.example.com`.

      Each database link should be created in the Oracle Streams administrator's schema. Also, each database link should connect to the Oracle Streams administrator at the destination database. See "Configuring Network Connectivity and Database Links" on page 1-17 for instructions.

   d. Set initialization parameters properly at the new spoke database. In this example, set initialization parameters properly at the `spoke3.example.com` database. See "Setting Initialization Parameters Relevant to Oracle Streams" on page 1-19 for instructions.

e. Configure the new spoke database to run in ARCHIVELOG mode. For a capture process to capture changes generated at a source database, the source database must be running in ARCHIVELOG mode. In this example, configure the spoke3.example.com database to run in ARCHIVELOG mode. See *Oracle Database Administrator's Guide* for information about configuring a database to run in ARCHIVELOG mode.

f. Ensure that the hub_dir directory objects exist at the hub.example.com database, and remove any files related to the previous configuration from it, including Data Pump export dump files and export log files.

2. Open SQL*Plus and connect to the spoke3.example.com database as the Oracle Streams administrator.

   See *Oracle Database 2 Day DBA* for more information about starting SQL*Plus.

3. Create a directory object to hold files that will be generated by the MAINTAIN_SCHEMAS procedure, including the Data Pump export dump file used for instantiation. The directory object can point to any accessible directory on the computer system. For example, the following statement creates a directory object named spoke3_dir that points to the /usr/spoke3_log_files directory:

```
CREATE DIRECTORY spoke3_dir AS '/usr/spoke3_log_files';
```

4. Stop the capture process at the hub database in the hub-and-spoke environment.

   Use the STOP_CAPTURE procedure in the DBMS_CAPTURE_ADM package to stop a capture process, or see *Oracle Database 2 Day + Data Replication and Integration Guide* for instructions about stopping a capture process using Oracle Enterprise Manager.

   In this example, stop the capture process at the hub.example.com database. The replicated database objects can remain open to changes while the capture process is stopped. These changes will be captured when the capture process is restarted.

5. In SQL*Plus, run the appropriate configuration procedure in the DBMS_STREAMS_ADM package at the hub database to add the new spoke database.

   In this example, complete the following steps:

   a. Open SQL*Plus and connect to the hub.example.com database as the Oracle Streams administrator.

   b. Run the MAINTAIN_SCHEMAS procedure to add the spoke3.example.com database to the Oracle Streams replication environment:

```
BEGIN
  DBMS_STREAMS_ADM.MAINTAIN_SCHEMAS(
    schema_names                 => 'hr',
    source_directory_object      => 'hub_dir',
    destination_directory_object => 'spoke3_dir',
    source_database              => 'hub.example.com',
    destination_database         => 'spoke3.example.com',
    capture_name                 => 'capture_hns',
    capture_queue_table          => 'source_hns_qt',
    capture_queue_name           => 'source_hns',
    propagation_name             => 'propagation_spoke3',
    apply_name                   => 'apply_spoke3',
    apply_queue_table            => 'destination_spoke3_qt',
    apply_queue_name             => 'destination_spoke3',
    bi_directional               => TRUE);
END;
/
```

The MAINTAIN_SCHEMAS procedure can take some time to run because it is performing many configuration tasks. Do not allow data manipulation language (DML) or data definition language (DDL) changes to the database objects in the specified schema at the destination database while the procedure is running. When the procedure completes, the new database objects are added to the environment, and the capture process that was stopped in Step 4 is restarted.

The parameter values specified in capture_name, capture_queue_table, and capture_queue_name must be the same as the values specified in the configuration procedure in the DBMS_STREAMS_ADM package that configured the replication environment. In this example, these parameter values match the ones specified in "Example That Configures Hub-and-Spoke Replication" on page 2-48.

When a configuration procedure is run, information about its progress is recorded in the following data dictionary views: DBA_RECOVERABLE_SCRIPT, DBA_RECOVERABLE_SCRIPT_PARAMS, DBA_RECOVERABLE_SCRIPT_BLOCKS, and DBA_RECOVERABLE_SCRIPT_ERRORS. If the procedure stops because it encounters an error, then see *Oracle Streams Replication Administrator's Guide* for instructions about using the RECOVER_OPERATION procedure in the DBMS_STREAMS_ADM package to recover from these errors.

6. Configure latest time conflict resolution for all of the tables in the hr schema at the spoke3.example.com database. This schema includes the countries, departments, employees, jobs, job_history, locations, and regions tables. See *Oracle Database 2 Day + Data Replication and Integration Guide* for instructions.

Adding Components Individually in Multiple Steps

This section describes adding Oracle Streams components separately to extend a replication environment. Oracle Streams components include queues, rules, rule sets, capture processes, synchronous captures, propagations, and apply processes.

This section contains these topics:

- Adding Replicated Objects to an Existing Single-Source Environment
- Adding a New Destination Database to a Single-Source Environment
- Adding Replicated Objects to an Existing Multiple-Source Environment
- Adding a New Database to an Existing Multiple-Source Environment

Note:

- When possible, it is usually easier to extend an Oracle Streams replication environment using either a single procedure or the Setup Streams Replication Wizard in Oracle Enterprise Manager. See "Adding Multiple Components Using a Single Procedure" on page 4-3 for instructions about using a single procedure and *Oracle Database 2 Day + Data Replication and Integration Guide* for instructions about using the wizard.
- The instructions in the following sections assume you will use the `DBMS_STREAMS_ADM` package to configure your Oracle Streams environment. If you use other packages, then extra steps might be necessary for each task.

Adding Replicated Objects to an Existing Single-Source Environment

You can add existing database objects to an existing single-source environment by adding the necessary rules to the appropriate capture processes, synchronous captures, propagations, and apply processes. Before creating or altering capture or propagation rules in a running Oracle Streams environment, ensure that any propagations or apply processes that will receive logical change records (LCRs) because of the new or altered rules are configured to handle these LCRs. That is, the propagations or apply processes should exist, and each one should be associated with rule sets that handle the LCRs appropriately. If these propagations and apply processes are not configured properly to handle these LCRs, then LCRs might be lost.

For example, suppose you want to add a table to an Oracle Streams replication environment that already captures, propagates, and applies changes to other tables. Assume that only one capture process or synchronous captures 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 rules to the following rule sets:

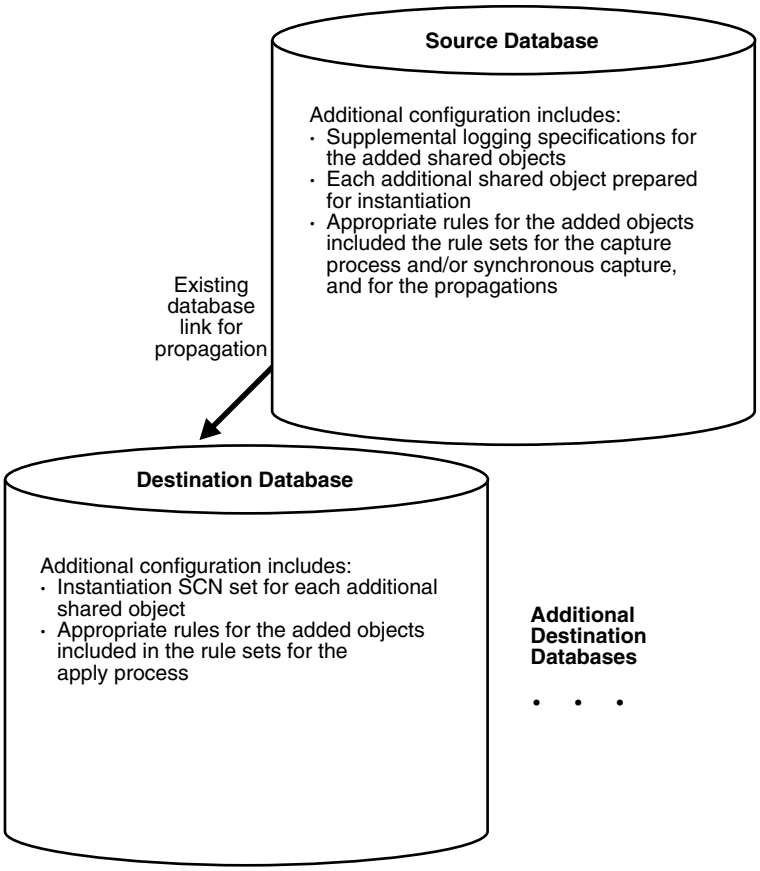
- The positive rule set for the apply process that will apply changes to the table
- The positive rule set for each propagation that will propagate changes to the table
- The positive rule set for the capture process or synchronous capture that will capture changes to the table

If you perform administrative steps in the wrong order, you can lose LCRs. For example, if you add the rule to a capture process 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 can be lost.

This example assumes that the replicated database objects are read-only at the destination databases. If the replicated database objects are read/write at the destination databases, then the replication environment will not stay synchronized because Oracle Streams is not configured to replicate the changes made to the replicated database objects at the destination databases.

Figure 4–1 shows the additional configuration steps that must be completed to add replicated database objects to a single-source Oracle Streams environment.

Figure 4–1 Example of Adding Replicated Objects to a Single-Source Environment



To avoid losing LCRs, complete the configuration in the following order:

1. At each source database where replicated database objects are being added, specify supplemental logging for the added replicated database objects. See "Specifying Supplemental Logging" on page 1-28 for instructions.
2. Either stop the capture process, one of the propagations, or the apply processes:
   - Use the `STOP_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package to stop a capture process, or see *Oracle Database 2 Day + Data Replication and Integration Guide* for instructions about stopping a capture process using Oracle Enterprise Manager.
   - Use the `STOP_PROPAGATION` procedure in the `DBMS_PROPAGATION_ADM` package to stop a propagation, or see *Oracle Database 2 Day + Data Replication and Integration Guide* for instructions about disabling a propagation using Oracle Enterprise Manager.
   - Use the `STOP_APPLY` procedure in the `DBMS_APPLY_ADM` package to stop an apply process, or see *Oracle Database 2 Day + Data Replication and Integration Guide* for instructions about stopping an apply process using Oracle Enterprise Manager.

   In general, it is best to stop the capture process so that messages do not accumulate in queues during the operation.

   Note: Synchronous captures cannot be stopped.

See Also: *Oracle Streams Concepts and Administration* for more information about completing these tasks with PL/SQL procedures

3. Add the relevant rules to the rule sets for the apply processes. To add rules to the rule set for an apply process, you can run one of the following procedures:
   - DBMS_STREAMS_ADM.ADD_TABLE_RULES
   - DBMS_STREAMS_ADM.ADD_SUBSET_RULES
   - DBMS_STREAMS_ADM.ADD_SCHEMA_RULES
   - DBMS_STREAMS_ADM.ADD_GLOBAL_RULES

   Excluding the ADD_SUBSET_RULES procedure, these procedures can add rules to the positive or negative rule set for an apply process. The ADD_SUBSET_RULES procedure can add rules only to the positive rule set for an apply process.
4. Add the relevant rules to the rule sets for the propagations. To add rules to the rule set for a propagation, you can run one of the following procedures:
   - DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES
   - DBMS_STREAMS_ADM.ADD_SUBSET_PROPAGATION_RULES
   - DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES
   - DBMS_STREAMS_ADM.ADD_GLOBAL_PROPAGATION_RULES

   Excluding the ADD_SUBSET_PROPAGATION_RULES procedure, these procedures can add rules to the positive or negative rule set for a propagation. The ADD_SUBSET_PROPAGATION_RULES procedure can add rules only to the positive rule set for a propagation.
5. Add the relevant rules to the rule sets used by the capture process or synchronous capture. To add rules to a rule set for an existing capture process, you can run one of the following procedures and specify the existing capture process:
   - DBMS_STREAMS_ADM.ADD_TABLE_RULES
   - DBMS_STREAMS_ADM.ADD_SUBSET_RULES
   - DBMS_STREAMS_ADM.ADD_SCHEMA_RULES
   - DBMS_STREAMS_ADM.ADD_GLOBAL_RULES

   Excluding the ADD_SUBSET_RULES procedure, these procedures can add rules to the positive or negative rule set for a capture process. The ADD_SUBSET_RULES procedure can add rules only to the positive rule set for a capture process.

   To add rules to a rule set for an existing synchronous capture, you can run one of the following procedures and specify the existing synchronous capture:
   - DBMS_STREAMS_ADM.ADD_TABLE_RULES
   - DBMS_STREAMS_ADM.ADD_SUBSET_RULES

   When you use a procedure in the DBMS_STREAMS_ADM package to add the capture process 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, if the capture process is a local capture process or a downstream capture process with a database link to the source database.

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 can be added to propagations and apply processes in the environment, but not to the capture process.
- You use a downstream capture process with no database link to the source database.

If you must prepare for instantiation manually, then see "Preparing Database Objects for Instantiation at a Source Database" on page 8-8 for instructions.

When you use a procedure in the DBMS_STREAMS_ADM package to add the synchronous capture rules, it automatically runs the PREPARE_SYNC_INSTANTIATION function in the DBMS_CAPTURE_ADM package for the specified table.

6. At each destination database, either instantiate, or set the instantiation SCN for, each database object you are adding to the Oracle Streams environment. If the database objects do not exist at a destination database, then instantiate them using export/import, transportable tablespaces, or RMAN. If the database objects already exist at a destination database, then set the instantiation SCNs for them manually.

   - To instantiate database objects using export/import, first export them at the source database. Next, import them at the destination database. See Chapter 8, "Instantiation and Oracle Streams Replication".

     Do not allow any changes to the database objects being exported while exporting these database objects at the source database. Do not allow changes to the database objects being imported while importing these database objects at the destination database.

     You can specify a more stringent degree of consistency by using an export parameter such as FLASHBACK_SCN or FLASHBACK_TIME.

   - 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 set the recursive parameter for this procedure to TRUE so that the instantiation SCN also is set for each schema at the destination database and for the tables owned by these schemas.

     If you run SET_SCHEMA_INSTANTIATION_SCN at a destination database, then set the recursive parameter for this procedure to TRUE so that the instantiation SCN also is set for each table in the schema.

If you set the `recursive` parameter to `TRUE` in the `SET_GLOBAL_INSTANTIATION_SCN` procedure or the `SET_SCHEMA_INSTANTIATION_SCN` procedure, then a database link from the destination database to the source database is required. This database link must have the same name as the global name of the source database and must be accessible to the user who executes the procedure. See "Setting Instantiation SCNs Using the DBMS_APPLY_ADM Package" on page 8-38 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 ensure that no rows are imported. Also, ensure that 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 8-37 for more information about performing a metadata export/import.

7. Start any Oracle Streams client you stopped in Step 2:
   - Use the `START_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package to start a capture process, or see *Oracle Database 2 Day + Data Replication and Integration Guide* for instructions about starting a capture process using Oracle Enterprise Manager.
   - Use the `START_PROPAGATION` procedure in the `DBMS_PROPAGATION_ADM` package start a propagation, or see *Oracle Database 2 Day + Data Replication and Integration Guide* for instructions about enabling a propagation using Oracle Enterprise Manager.
   - Use the `START_APPLY` procedure in the `DBMS_APPLY_ADM` package to start an apply process, or see *Oracle Database 2 Day + Data Replication and Integration Guide* for instructions about starting an apply process using Oracle Enterprise Manager.

   See Also: *Oracle Streams Concepts and Administration* for more information about completing these tasks using PL/SQL procedures

You must stop the capture process, disable one of the propagation jobs, or stop the apply process in Step 2 to ensure that the table or schema is instantiated before the first LCR resulting from the added rule(s) reaches the apply process. Otherwise, LCRs could be lost or could result in apply errors, depending on whether the apply process 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 7 before Step 6 to reduce the amount of time that an Oracle Streams process or propagation job is stopped.

See Also: *Oracle Streams Extended Examples* for a detailed example that adds objects to an existing single-source environment

Adding a New Destination Database to a Single-Source Environment

You can 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 send changes to the new

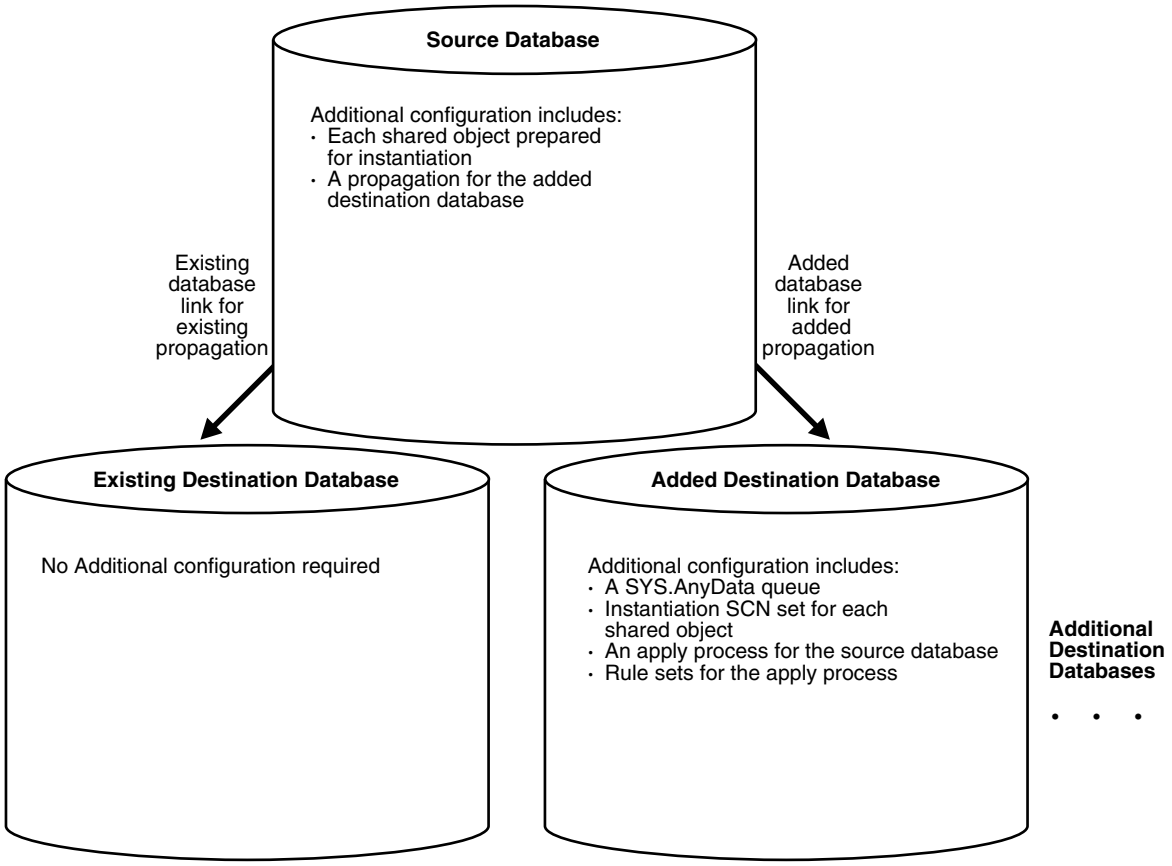
destination database. You might also need to add rules to existing propagations in the stream that send changes to the new destination database.

As in the example that describes "Adding Replicated Objects to an Existing Single-Source Environment" on page 4-11, before creating or altering propagation rules in a running Oracle Streams replication environment, ensure that any propagations or apply processes that will receive logical change records (LCRs) because of the new or altered rules are configured to handle these LCRs. Otherwise, LCRs might be lost.

This example assumes that the replicated database objects are read-only at the destination databases. If the replicated database objects are read/write at the destination databases, then the replication environment will not stay synchronized because Oracle Streams is not configured to replicate the changes made to the replicated database objects at the destination databases.

Figure 4–2 shows the additional configuration steps that must be completed to add a destination database to a single-source Oracle Streams environment.

Figure 4–2 Example of Adding a Destination to a Single-Source Environment



To avoid losing LCRs, you should complete the configuration in the following order:

1. Complete the necessary tasks to prepare each database in your environment for Oracle Streams. See "Tasks to Complete Before Configuring Oracle Streams Replication" on page 1-13.

   Some of these tasks might not be required at certain databases.

2. Create any necessary `ANYDATA` queues that do not already exist at the destination database. When you create an apply process, you associate the apply process with

a specific ANYDATA queue. See "Creating an ANYDATA Queue" on page 6-1 for instructions.

3. Create one or more apply processes at the new destination database to apply the changes from its source database. Ensure that each apply process uses rule sets that are appropriate for applying changes. Do not start any of the apply processes at the new database. See Chapter 7, "Configuring Implicit Apply" 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. Starting the apply processes might lead to incorrect data and errors.

4. Configure any necessary propagations to send changes from the source databases to the new destination database. Ensure that each propagation uses rule sets that are appropriate for propagating changes. See "Creating Oracle Streams Propagations Between ANYDATA Queues" on page 6-3.

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.

   If you are using one or more capture processes, then 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.

   If you are using one or more synchronous captures, then run the PREPARE_SYNC_INSTANTIATION function in the DBMS_CAPTURE_ADM package for the specified table.

   See "Preparing Database Objects for Instantiation at a Source Database" on page 8-8.

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, transportable tablespaces, or RMAN. If the database objects exist at the new destination database, then set the instantiation SCNs for them.

   - To instantiate database objects using export/import, first export them at the source database. Next, import them at the destination database. See Chapter 8, "Instantiation and Oracle Streams Replication".

     Do not allow any changes to the database objects being exported while exporting these database objects at the source database. Do not allow changes to the database objects being imported while importing these database objects at the destination database.

     You can specify a more stringent degree of consistency by using an export parameter such as FLASHBACK_SCN or FLASHBACK_TIME.

   - 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 replicated database 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 set the `recursive` parameter for this procedure to `TRUE` so that the instantiation SCN also is set for each schema at the destination database and for the tables owned by these schemas.

If you run `SET_SCHEMA_INSTANTIATION_SCN` at a destination database, then set the `recursive` parameter for this procedure to `TRUE` so that the instantiation SCN also is set for each table in the schema.

If you set the `recursive` parameter to `TRUE` in the `SET_GLOBAL_INSTANTIATION_SCN` procedure or the `SET_SCHEMA_INSTANTIATION_SCN` procedure, then a database link from the destination database to the source database is required. This database link must have the same name as the global name of the source database and must be accessible to the user who executes the procedure. See "Setting Instantiation SCNs Using the DBMS_APPLY_ADM Package" on page 8-38 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 ensure that no rows are imported. Also, ensure that the replicated database 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 8-37 for more information about performing a metadata export/import.

7. Start the apply processes you created in Step 3 using the `START_APPLY` procedure in the `DBMS_APPLY_ADM` package, or see *Oracle Database 2 Day + Data Replication and Integration Guide* for instructions about starting an apply process using Oracle Enterprise Manager.

See Also: *Oracle Streams Extended Examples* for detailed example that adds a database to an existing single-source environment

Adding Replicated Objects to an Existing Multiple-Source Environment

You can add existing database objects to an existing multiple-source environment by adding the necessary rules to the appropriate capture processes, synchronous captures, propagations, and apply processes.

This example uses the following terms:

- **Populated database:** A database that already contains the replicated 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 might 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 replicated database objects before they are added to the multiple-source environment. You instantiate the replicated database objects at an import database by performing an import of

these database objects. You might 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 Oracle Streams replication environment, ensure that any propagations or apply processes that will receive logical change records (LCRs) because of the new or altered rules are configured to handle these LCRs. That is, the propagations or apply processes should exist, and each one should be associated with rule sets that handle the LCRs appropriately. If these propagations and apply processes are not configured properly to handle these LCRs, then LCRs can be lost.

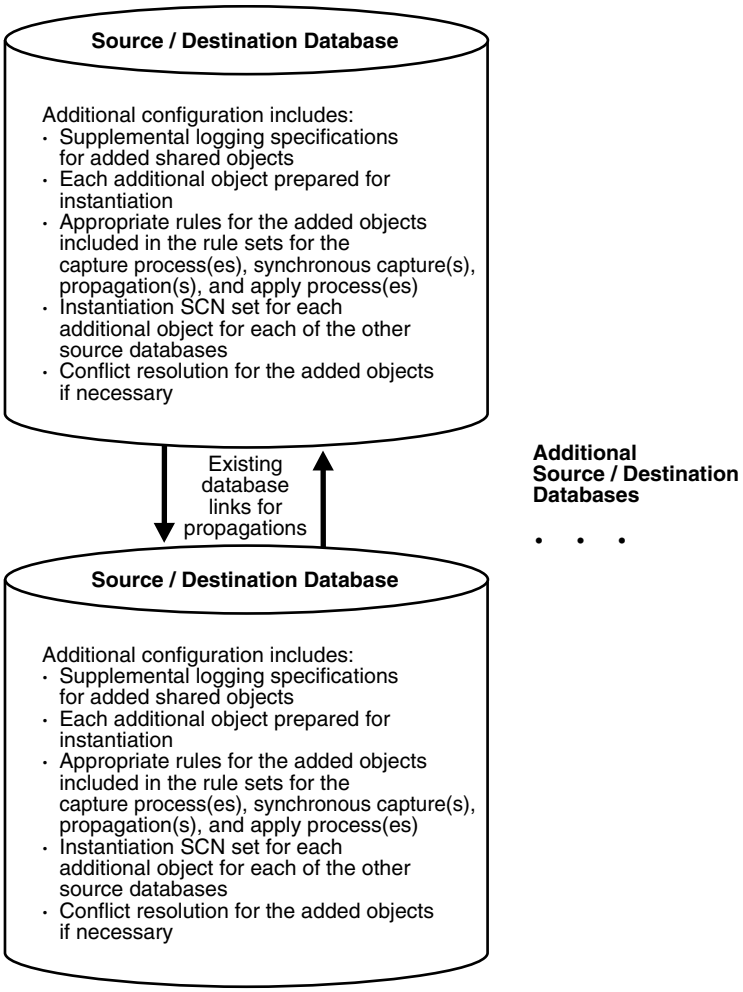
For example, suppose you want to add a new table to an Oracle Streams replication environment that already captures, propagates, and applies changes to other tables. Assume multiple capture processes or synchronous captures in the environment will capture changes to this table, and multiple apply processes will apply changes to this table. In this case, you must add one or more table rules to the following rule sets:

- The positive rule set for each apply process that will apply changes to the table
- The positive rule set for each propagation that will propagate changes to the table
- The positive rule set for each capture process or synchronous capture that will capture changes to the table

If you perform administrative steps in the wrong order, then you can lose LCRs. For example, if you add the rule to a capture process 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 can be lost.

Figure 4–3 shows the additional configuration steps that must be completed to add replicated database objects to a multiple-source Oracle Streams environment.

Figure 4–3 Example of Adding Replicated Objects to a Multiple-Source Environment



When there are multiple source databases in an Oracle Streams replication environment, change cycling is possible. Change cycling happens when a change is sent back to the database where it originated. Typically, you should avoid change cycling. Before you configure your replication environment, see Chapter 10, "Oracle Streams Tags", and ensure that you configure the replication environment to avoid change cycling.

To avoid losing LCRs, 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" on page 1-28 for instructions.
2. Either stop all of the capture processes that will capture changes to the added objects, stop all of the propagations that will propagate changes to the added objects, or stop all of the apply process that will apply changes to the added objects:
   - Use the `STOP_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package to stop a capture process, or see *Oracle Database 2 Day + Data Replication and Integration Guide* for instructions about stopping a capture process using Oracle Enterprise Manager.

- Use the `STOP_PROPAGATION` procedure in the `DBMS_PROPAGATION_ADM` package to stop a propagation, or see *Oracle Database 2 Day + Data Replication and Integration Guide* for instructions about disabling a propagation using Oracle Enterprise Manager.
- Use the `STOP_APPLY` procedure in the `DBMS_APPLY_ADM` package to stop an apply process, or see *Oracle Database 2 Day + Data Replication and Integration Guide* for instructions about stopping an apply process using Oracle Enterprise Manager.

In general, it is best to stop the capture process so that messages do not accumulate in queues during the operation.

Note: Synchronous captures cannot be stopped.

See Also: *Oracle Streams Concepts and Administration* for more information about completing these tasks using PL/SQL procedures

3. Add the relevant rules to the rule sets for the apply processes that will apply changes to the added objects. To add rules to the rule set for an apply process, you can run one of the following procedures:
   - `DBMS_STREAMS_ADM.ADD_TABLE_RULES`
   - `DBMS_STREAMS_ADM.ADD_SUBSET_RULES`
   - `DBMS_STREAMS_ADM.ADD_SCHEMA_RULES`
   - `DBMS_STREAMS_ADM.ADD_GLOBAL_RULES`

   Excluding the `ADD_SUBSET_RULES` procedure, these procedures can add rules to the positive or negative rule set for an apply process. The `ADD_SUBSET_RULES` procedure can add rules only to the positive rule set for an apply process.
4. Add the relevant rules to the rule sets for the propagations that will propagate changes to the added objects. To add rules to the rule set for a propagation, you can run one of the following procedures:
   - `DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES`
   - `DBMS_STREAMS_ADM.ADD_SUBSET_PROPAGATION_RULES`
   - `DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES`
   - `DBMS_STREAMS_ADM.ADD_GLOBAL_PROPAGATION_RULES`

   Excluding the `ADD_SUBSET_PROPAGATION_RULES` procedure, these procedures can add rules to the positive or negative rule set for a propagation. The `ADD_SUBSET_PROPAGATION_RULES` procedure can add rules only to the positive rule set for a propagation.
5. Add the relevant rules to the rule sets used by each capture process or synchronous capture that will capture changes to the added objects. To add rules to a rule set for an existing capture process, you can run one of the following procedures and specify the existing capture process:
   - `DBMS_STREAMS_ADM.ADD_TABLE_RULES`
   - `DBMS_STREAMS_ADM.ADD_SUBSET_RULES`

- DBMS_STREAMS_ADM.ADD_SCHEMA_RULES
- DBMS_STREAMS_ADM.ADD_GLOBAL_RULES

Excluding the ADD_SUBSET_RULES procedure, these procedures can add rules to the positive or negative rule set for a capture process. The ADD_SUBSET_RULES procedure can add rules only to the positive rule set for a capture process.

To add rules to a rule set for an existing synchronous capture, you can run one of the following procedures and specify the existing synchronous capture:

- DBMS_STREAMS_ADM.ADD_TABLE_RULES
- DBMS_STREAMS_ADM.ADD_SUBSET_RULES

When you use a procedure in the DBMS_STREAMS_ADM package to add the capture process 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, if the capture process is a local capture process or a downstream capture process with a database link to the source database.

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 can be added to propagations and apply processes in the environment, but not to the capture process.
- You use a downstream capture process with no database link to the source database.

If you must prepare for instantiation manually, then see "Preparing Database Objects for Instantiation at a Source Database" on page 8-8 for instructions.

When you use a procedure in the DBMS_STREAMS_ADM package to add the synchronous capture rules, it automatically runs the PREPARE_SYNC_INSTANTIATION function in the DBMS_CAPTURE_ADM package for the specified table.

After completing these steps, complete the steps in each of the following sections that apply to your environment. You might 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 Replicated Objects" on page 4-22. These steps are required only if your environment has multiple populated databases.
- For each import database, complete the steps in "Adding Replicated Objects to Import Databases in an Existing Environment" on page 4-23.

Configuring Populated Databases When Adding Replicated Objects

After completing the steps in "Adding Replicated Objects to an Existing Multiple-Source Environment" on page 4-18, complete the following steps for each populated database if your environment has multiple populated databases:

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 database object in one of the following ways:

   - Perform a metadata only export of the added database 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. Ensure that no rows are imported. Also, ensure that the replicated database 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 replicating DML changes only, then table level export/import is sufficient. If you are replicating DDL changes also, then additional considerations apply. See "Setting Instantiation SCNs Using Export/Import" on page 8-37 for more information about performing a metadata export/import.

   - Set the instantiation SCNs manually for the added objects at each of the other populated databases. Ensure that every added database 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 8-38 for instructions.

Adding Replicated Objects to Import Databases in an Existing Environment

After completing the steps in "Adding Replicated Objects to an Existing Multiple-Source Environment" on page 4-18, 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 database objects at all of the other databases in the environment that will be a destination database of the import database. In this case, the import database will be the source database for these destination databases. The databases where you set the instantiation SCNs might 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 replicated 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 replicated 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 Using the DBMS_APPLY_ADM Package" on page 8-38 for instructions.

   Because you run these procedures before any tables are instantiated at the import databases, and because the local capture processes or synchronous captures are configured already for these import databases, you will not need to run the `SET_`

TABLE_INSTANTIATION_SCN procedure for each table created during instantiation. Instantiation SCNs will be set automatically for these tables at all of the other databases in the environment that will be destination databases of the import database.

3. At the export database you chose in Step 1, perform an export of the replicated database objects. Next, perform an import of the replicated database objects at each import database. See Chapter 8, "Instantiation and Oracle Streams Replication" and *Oracle Database Utilities* for information about using export/import.

   Do not allow any changes to the database objects being exported while exporting these database objects at the source database. Do not allow changes to the database objects being imported while importing these database objects at the destination database.

   You can specify a more stringent degree of consistency by using an export parameter such as FLASHBACK_SCN or FLASHBACK_TIME.

4. For each populated database, except for the export database, set the instantiation SCNs for the added database 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:

   - Perform a metadata only export of the added database 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 database object at the import database is consistent with the populated database at the time of the export.

     If you are replicating DML changes only, then table level export/import is sufficient. If you are replicating DDL changes also, then additional considerations apply. See "Setting Instantiation SCNs Using Export/Import" on page 8-37 for more information about performing a metadata export/import.

   - Set the instantiation SCNs manually for the added objects at each import database. Ensure that 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 8-38 for instructions.

Finish Adding Objects to a Multiple-Source Environment Configuration

Before completing the configuration, you should have completed the following tasks:

- "Adding Replicated Objects to an Existing Multiple-Source Environment" on page 4-18
- "Configuring Populated Databases When Adding Replicated Objects" on page 4-22, if your environment has multiple populated databases
- "Adding Replicated Objects to Import Databases in an Existing Environment" on page 4-23, if your environment had import databases

When all of the previous configuration steps are finished, complete the following steps:

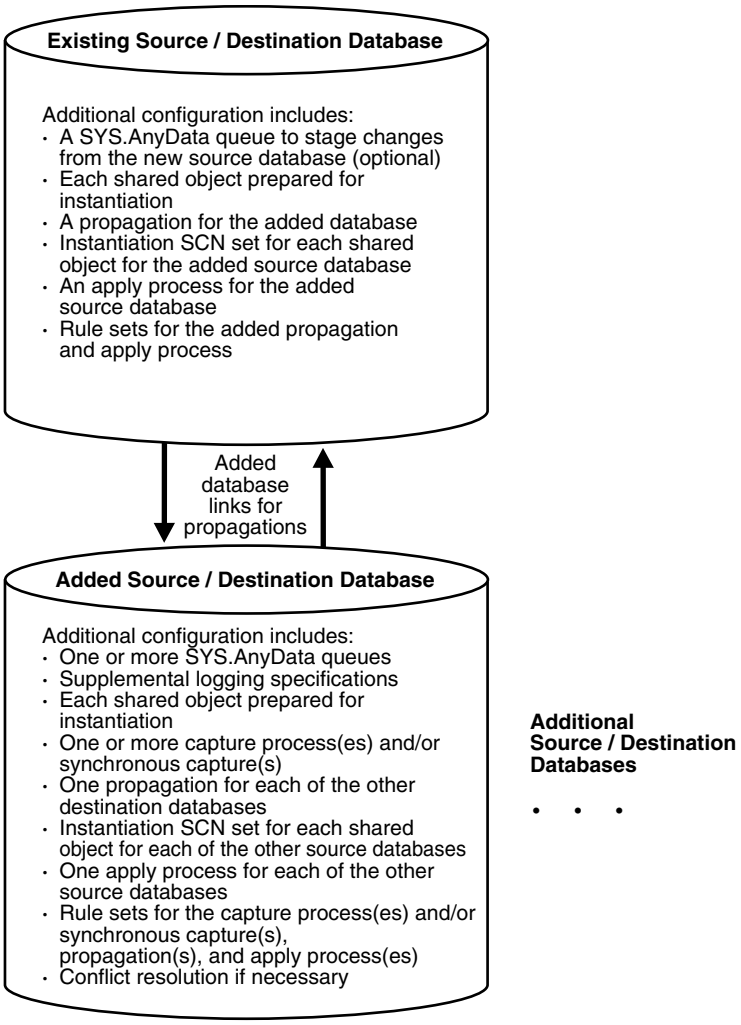
1. At each database, configure conflict resolution for the added database objects if conflicts are possible. See Chapter 9, "Oracle Streams Conflict Resolution" for instructions.
2. Start each Oracle Streams client you stopped in Step 2 on page 4-20 in "Adding Replicated Objects to an Existing Multiple-Source Environment":
   - Use the `START_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package to start a capture process, or see *Oracle Database 2 Day + Data Replication and Integration Guide* for instructions about starting a capture process using Oracle Enterprise Manager.
   - Use the `START_PROPAGATION` procedure in the `DBMS_PROPAGATION_ADM` package start a propagation, or see *Oracle Database 2 Day + Data Replication and Integration Guide* for instructions about enabling a propagation using Oracle Enterprise Manager.
   - Use the `START_APPLY` procedure in the `DBMS_APPLY_ADM` package to start an apply process, or see *Oracle Database 2 Day + Data Replication and Integration Guide* for instructions about starting an apply process using Oracle Enterprise Manager.

     See Also: *Oracle Streams Concepts and Administration* for more information about completing these tasks using the PL/SQL procedures

Adding a New Database to an Existing Multiple-Source Environment

Figure 4–4 shows the additional configuration steps that must be completed to add a source/destination database to a multiple-source Oracle Streams environment.

Figure 4–4 Example of Adding a Database to a Multiple-Source Environment



When there are multiple source databases in an Oracle Streams replication environment, change cycling is possible. Change cycling happens when a change is sent back to the database where it originated. Typically, you should avoid change cycling. Before you configure your replication environment, see Chapter 10, "Oracle Streams Tags", and ensure that you configure the replication environment to avoid change cycling.

Complete the following steps to add a new source/destination database to an existing multiple-source Oracle Streams replication environment:

Note: Ensure that no changes are made to the database objects being replicated at the database you are adding to the Oracle Streams replication environment until the instantiation at the database is complete.

---

1. Complete the necessary tasks to prepare each database in your environment for Oracle Streams. See "Tasks to Complete Before Configuring Oracle Streams Replication" on page 1-13.

   Some of these tasks might not be required at certain databases.

2. Create any necessary ANYDATA queues that do not already exist. When you create a capture process, synchronous capture, or apply process, you associate the process with a specific ANYDATA queue. When you create a propagation, you associate it with a specific source queue and destination queue. See "Creating an ANYDATA Queue" on page 6-1 for instructions.

3. Create one or more apply processes at the new database to apply the changes from its source databases. Ensure that each apply process uses rule sets that are appropriate for applying changes. Do not start any apply process at the new database. See Chapter 7, "Configuring Implicit Apply" 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. Starting the apply processes might 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. Ensure that each apply process uses rule sets that are appropriate for applying changes. Do not start any of these new apply processes. See Chapter 7, "Configuring Implicit Apply" for instructions.

5. Configure propagations at the databases that will be source databases of the new database to send changes to the new database. Ensure that each propagation uses rule sets that are appropriate for propagating changes. See "Creating Oracle Streams Propagations Between ANYDATA Queues" on page 6-3.

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. Ensure that each propagation uses rule sets that are appropriate for propagating changes. See "Creating Oracle Streams Propagations Between ANYDATA Queues" on page 6-3.

7. If the new database will be a source database, and the replicated database objects already exist at the new database, then specify any necessary supplemental logging for the replicated database objects at the new database. See "Specifying Supplemental Logging" on page 1-28 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.

   If you are using one or more capture processes, then 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.

   If you are using one or more synchronous captures, then run the PREPARE_TABLE_INSTANTIATION procedure in the DBMS_CAPTURE_ADM package for the specified table.

   See "Preparing Database Objects for Instantiation at a Source Database" on page 8-8 for instructions.

9. If the new database will be a source database, then create one or more capture processes or synchronous captures to capture the relevant changes. See Chapter 5, "Configuring Implicit Capture" for instructions. If you plan to use capture processes, then Oracle recommends that you use only one capture process for each source database.

When you use a procedure in the DBMS_STREAMS_ADM package to add the capture process 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, if the capture process is a local capture process or a downstream capture process with a database link to the source database.

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 rules.
- You use an existing capture process and do not add capture process rules for any replicated database object.
- You use a downstream capture process with no database link to the source database.

If you must prepare for instantiation manually, then see "Preparing Database Objects for Instantiation at a Source Database" on page 8-8 for instructions.

When you use a procedure in the DBMS_STREAMS_ADM package to add the synchronous capture rules, it automatically runs the PREPARE_SYNC_INSTANTIATION function in the DBMS_CAPTURE_ADM package for the specified table.

10. If the new database will be a source database, then start any capture process you created in Step 9 using the START_CAPTURE procedure in the DBMS_CAPTURE_ADM package, or see *Oracle Database 2 Day + Data Replication and Integration Guide* for instructions about starting a capture process using Oracle Enterprise Manager.

After completing these steps, complete the steps in the appropriate section:

- If the objects that are to be replicated with the new database already exist at the new database, then complete the steps in "Configuring Databases If the Replicated Objects Already Exist at the New Database" on page 4-28.
- If the objects that are to be replicated with the new database do not already exist at the new database, complete the steps in "Adding Replicated Objects to a New Database" on page 4-29.

Configuring Databases If the Replicated Objects Already Exist at the New Database

After completing the steps in "Adding a New Database to an Existing Multiple-Source Environment" on page 4-25, complete the following steps if the objects that are to be replicated 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:

   - Perform a metadata only export of the replicated database 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. Ensure that no rows are imported. In this case, ensure that the replicated database objects at the new database are consistent with the source database at the time of the export.

If you are replicating DML changes only, then table level export/import is sufficient. If you are replicating DDL changes also, then additional considerations apply. See "Setting Instantiation SCNs Using Export/Import" on page 8-37 for more information about performing a metadata export/import.

- Set the instantiation SCNs manually at the new database for the replicated database objects. Ensure that the replicated database 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 8-38 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:

   - Perform a metadata only export at the new database and import the metadata at each destination database. Ensure that 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 replicated database objects at each destination database are consistent with the new database at the time of the export.

     If you are replicating DML changes only, then table level export/import is sufficient. If you are replicating DDL changes also, then additional considerations apply. See "Setting Instantiation SCNs Using Export/Import" on page 8-37 for more information about performing a metadata export/import.

   - Set the instantiation SCNs manually at each destination database for the replicated database objects. Ensure that the replicated database 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 8-38 for instructions.

3. At the new database, configure conflict resolution if conflicts are possible. See Chapter 9, "Oracle Streams Conflict Resolution" for instructions.

4. Start the apply processes that you created at the new database in Step 3 on page 4-27 using the `START_APPLY` procedure in the `DBMS_APPLY_ADM` package, or see *Oracle Database 2 Day + Data Replication and Integration Guide* for instructions about starting an apply process using Oracle Enterprise Manager.

5. Start the apply processes that you created at each of the other destination databases in Step 4 on page 4-27. If the new database is not a source database, then do not complete this step.

Adding Replicated Objects to a New Database

After completing the steps in "Adding a New Database to an Existing Multiple-Source Environment" on page 4-25, complete the following steps if the database 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 replicated 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 replicated 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 Using the DBMS_APPLY_ADM Package" on page 8-38 for instructions.

Because you run these procedures before any tables are instantiated at the new database, and because the local capture process or synchronous capture is configured already at the new database, you will not need to run the `SET_TABLE_INSTANTIATION_SCN` procedure for each table created during instantiation. Instantiation SCNs will be set automatically for these tables at all of the other databases in the environment that will be destination databases of the new database.

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 replicated database objects at the new database using export/import. First, perform an export of the replicated database objects. Next, perform an import of the replicated database objects at the new database. See Chapter 8, "Instantiation and Oracle Streams Replication" and *Oracle Database Utilities* for information about using export/import.

   Do not allow any changes to the database objects being exported while exporting these database objects at the source database. Do not allow changes to the database objects being imported while importing these database objects at the destination database.

   You can specify a more stringent degree of consistency by using an export parameter such as `FLASHBACK_SCN` or `FLASHBACK_TIME`.

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:

   - 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 replicated database objects at the new database are consistent with the source database at the time of the export.

     If you are replicating DML changes only, then table level export/import is sufficient. If you are replicating DDL changes also, then additional considerations apply. See "Setting Instantiation SCNs Using Export/Import" on page 8-37 for more information about performing a metadata export/import.

- Set the instantiation SCNs manually at the new database for the replicated database objects. Ensure that the replicated database 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 8-38 for instructions.

4. At the new database, configure conflict resolution if conflicts are possible. See Chapter 9, "Oracle Streams Conflict Resolution" for instructions.
5. Start the apply processes that you created in Step 3 on page 4-27 at the new database using the `START_APPLY` procedure in the `DBMS_APPLY_ADM` package, or see *Oracle Database 2 Day + Data Replication and Integration Guide* for instructions about starting an apply process using Oracle Enterprise Manager.
6. Start the apply processes that you created in Step 4 on page 4-27 at each of the other destination databases. If the new database is not a source database, then do not complete this step.

5

Configuring Implicit Capture

Implicit capture means that a capture process or a synchronous capture captures and enqueues database changes automatically. A capture process captures changes in the redo log, while a synchronous capture captures data manipulation language (DML) changes with an internal mechanism. Both capture processes and synchronous captures reformat the captured changes into logical change records (LCRs) and enqueue the LCRs into an `ANYDATA` queue.

The following topics describe configuring implicit capture:

- Configuring a Capture Process
- Configuring Synchronous Capture

Each task described in this chapter should be completed by an Oracle Streams administrator that has been granted the appropriate privileges, unless specified otherwise.

See Also:

- *Oracle Streams Concepts and Administration* for more information about implicit capture
- "Configuring an Oracle Streams Administrator on All Databases" on page 1-13

Configuring a Capture Process

You can create a capture process that captures changes either locally at the source database or remotely at a downstream database. A downstream capture process runs on a downstream database, and redo data from the source database is copied to the downstream database. A downstream capture process captures changes in the copied redo data at the downstream database.

You can use any of the following procedures to create a local capture process:

- `DBMS_STREAMS_ADM.ADD_TABLE_RULES`
- `DBMS_STREAMS_ADM.ADD_SUBSET_RULES`
- `DBMS_STREAMS_ADM.ADD_SCHEMA_RULES`
- `DBMS_STREAMS_ADM.ADD_GLOBAL_RULES`
- `DBMS_CAPTURE_ADM.CREATE_CAPTURE`

Each of the procedures in the `DBMS_STREAMS_ADM` package creates a capture process with the specified name if it does not already exist, creates either a positive rule set or

negative rule set for the capture process if the capture process does not have such a rule set, and can add table rules, schema rules, or global rules to the rule set.

The `CREATE_CAPTURE` procedure creates a capture process, but does not create a rule set or rules for the capture process. However, the `CREATE_CAPTURE` procedure enables you to specify an existing rule set to associate with the capture process, either as a positive or a negative rule set, a first SCN, and a start SCN for the capture process. Also, to create a capture process that performs downstream capture, you must use the `CREATE_CAPTURE` procedure.

The following sections describe configuring a capture process:

- Preparing to Configure a Capture Process
- Configuring a Local Capture Process
- Configuring a Downstream Capture Process
- After Configuring a Capture Process

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

Note:

- You can configure an entire Oracle Streams environment, including capture processes, using procedures in the `DBMS_STREAMS_ADM` package or Oracle Enterprise Manager. See Chapter 2, "Simple Oracle Streams Replication Configuration".
- After creating a capture process, avoid changing the `DBID` or global name of the source database for the capture process. If you change either the `DBID` or global name of the source database, then the capture process must be dropped and re-created. See "Changing the DBID or Global Name of a Source Database" on page 12-23.
- To configure downstream capture, the source database must be an Oracle Database 10*g* Release 1 or later database.

Preparing to Configure a Capture Process

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

- Complete the following tasks in "Tasks to Complete Before Configuring Oracle Streams Replication" on page 1-13.
  - "Configuring an Oracle Streams Administrator on All Databases" on page 1-13
  - "Configuring Network Connectivity and Database Links" on page 1-17

- "Ensuring That Each Source Database Is In ARCHIVELOG Mode" on page 1-19
- "Setting Initialization Parameters Relevant to Oracle Streams" on page 1-19
- "Configuring the Oracle Streams Pool" on page 1-25
- "Specifying Supplemental Logging" on page 1-28

- If you plan to create a real-time or an archived-log downstream capture process that uses redo transport services to transfer archived redo log files to the downstream database automatically, then complete the steps in "Configuring Log File Transfer to a Downstream Capture Database" on page 1-33.
- If you plan to create a real-time downstream capture process, then complete the steps in "Adding Standby Redo Logs for Real-Time Downstream Capture" on page 1-36.
- Create an `ANYDATA` queue to associate with the capture process, if one does not exist. See "Creating an ANYDATA Queue" on page 6-1 for instructions. The examples in this chapter assume that the queue used by the capture process is `strmadmin.streams_queue`. Create the queue on the same database that will run the capture process.

Configuring a Local Capture Process

The following sections describe using the `DBMS_STREAMS_ADM` package and the `DBMS_CAPTURE_ADM` package to create a local capture process.

This section contains the following examples:

- Configuring a Local Capture Process Using DBMS_STREAMS_ADM
- Configuring a Local Capture Process Using DBMS_CAPTURE_ADM
- Configuring a Local Capture Process with Non-NULL Start SCN

Configuring a Local Capture Process Using DBMS_STREAMS_ADM

To configure a local capture process using the `DBMS_STREAMS_ADM` package, complete the following steps:

1. Complete the tasks in "Preparing to Configure a Capture Process" on page 5-2.
2. In SQL*Plus, connect to the source database as the Oracle Streams administrator.

   See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.
3. Run the `ADD_TABLE_RULES` procedure in the `DBMS_STREAMS_ADM` package to create a local capture process:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name         => 'hr.employees',
    streams_type       => 'capture',
    streams_name       => 'strm01_capture',
    queue_name         => 'strmadmin.streams_queue',
    include_dml        => TRUE,
    include_ddl        => TRUE,
    include_tagged_lcr => FALSE,
    source_database    => NULL,
    inclusion_rule     => TRUE);
END;
```

```
/
```

Running this procedure performs the following actions:

- Creates a capture process named `strm01_capture`. The capture process is created only if it does not already exist. If a new capture process is created, then this procedure also sets the start SCN to the point in time of creation.
- Associates the capture process with the existing queue `strmadmin.streams_queue`.
- Creates a positive rule set and associates it with the capture process, if the capture process does not have a positive rule set. The rule set is a positive rule set because the `inclusion_rule` parameter is set to `TRUE`. The rule set uses the `SYS.STREAMS$_EVALUATION_CONTEXT` evaluation context. The rule set name is system generated.
- Creates two rules. One rule evaluates to `TRUE` for data manipulation language (DML) changes to the `hr.employees` table, and the other rule evaluates to `TRUE` for data definition language (DDL) changes to the `hr.employees` table. The rule names are system generated.
- Adds the two rules to the positive rule set associated with the capture process. The rules are added to the positive rule set because the `inclusion_rule` parameter is set to `TRUE`.
- Specifies that the capture process captures a change in the redo log only if the change has a `NULL` tag, because the `include_tagged_lcr` parameter is set to `FALSE`. This behavior is accomplished through the system-created rules for the capture process.
- Creates a capture process that captures local changes to the source database because the `source_database` parameter is set to `NULL`. For a local capture process, you can also specify the global name of the local database for this parameter.
- Prepares the `hr.employees` table for instantiation.
- Enables supplemental logging for any primary key, unique key, bitmap index, and foreign key columns in the `hr.employees` table.

4. If necessary, complete the steps described in "After Configuring a Capture Process" on page 5-17.

See Also:

- *Oracle Streams Concepts and Administration* for more information about rules
- Chapter 10, "Oracle Streams Tags"

Configuring a Local Capture Process Using DBMS_CAPTURE_ADM

To configure a local capture process using the `DBMS_CAPTURE_ADM` package, complete the following steps:

1. Complete the tasks in "Preparing to Configure a Capture Process" on page 5-2.
2. In SQL*Plus, connect to the source database as the Oracle Streams administrator.

   See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.

3. Create the rule set that will be used by the capture process if it does not exist. In this example, assume that the rule set is `strmadmin.strm01_rule_set`. Optionally, you can also add rules to the rule set. See *Oracle Streams Concepts and Administration* for instructions.

4. Run the `CREATE_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package to create a local capture process:

```
BEGIN
  DBMS_CAPTURE_ADM.CREATE_CAPTURE(
    queue_name         => 'strmadmin.streams_queue',
    capture_name       => 'strm02_capture',
    rule_set_name      => 'strmadmin.strm01_rule_set',
    start_scn          => NULL,
    source_database    => NULL,
    first_scn          => NULL);
END;
/
```

   Running this procedure performs the following actions:

   - Creates a capture process named `strm02_capture`. A capture process with the same name must not exist.
   - Associates the capture process with the existing queue `strmadmin.streams_queue`.
   - Associates the capture process with the existing rule set `strmadmin.strm01_rule_set`. This rule set is the positive rule set for the capture process.
   - Creates a capture process that captures local changes to the source database because the `source_database` parameter is set to `NULL`. For a local capture process, you can also specify the global name of the local database for this parameter.
   - Specifies that the Oracle database determines the start SCN and first SCN for the capture process because both the `start_scn` parameter and the `first_scn` parameter are set to `NULL`.
   - If no other capture processes that capture local changes are running on the local database, then the `BUILD` procedure in the `DBMS_CAPTURE_ADM` package is run automatically. Running this procedure extracts the data dictionary to the redo log, and a LogMiner data dictionary is created when the capture process is started for the first time.

5. If necessary, complete the steps described in "After Configuring a Capture Process" on page 5-17.

   See Also: *Oracle Streams Concepts and Administration* for more information about rules

Configuring a Local Capture Process with Non-NULL Start SCN

This example runs the `CREATE_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package to create a local capture process with a start SCN set to `223525`. This example assumes that there is at least one local capture process at the database, and that this capture process has taken at least one checkpoint. You can always specify a start SCN for a new capture process that is equal to or greater than the current SCN of the source database. To specify a start SCN that is lower than the current SCN of the database, the specified start SCN must be higher than the lowest first SCN for an existing local

capture process that has been started successfully at least once and has taken at least one checkpoint.

You can determine the first SCN for existing capture processes, and whether these capture processes have taken a checkpoint, by running the following query:

```
SELECT CAPTURE_NAME, FIRST_SCN, MAX_CHECKPOINT_SCN FROM DBA_CAPTURE;
```

Your output looks similar to the following:

```
CAPTURE_NAME                   FIRST_SCN MAX_CHECKPOINT_SCN
------------------------------ ---------- ------------------
CAPTURE_SIMP                      223522             230825
```

These results show that the `capture_simp` capture process has a first SCN of `223522`. Also, this capture process has taken a checkpoint because the `MAX_CHECKPOINT_SCN` value is non-`NULL`. Therefore, the start SCN for the new capture process can be set to `223522` or higher.

To configure a local capture process with a non-`NULL` start SCN, complete the following steps:

1. Complete the tasks in "Preparing to Configure a Capture Process" on page 5-2.
2. In SQL*Plus, connect to the source database as the Oracle Streams administrator.

   See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.
3. Create the rule set that will be used by the capture process if it does not exist. In this example, assume that the rule set is `strmadmin.strm01_rule_set`. Optionally, you can also add rules to the rule set. See *Oracle Streams Concepts and Administration* for instructions.
4. Run the following procedure to create the capture process:

   ```
   BEGIN
     DBMS_CAPTURE_ADM.CREATE_CAPTURE(
       queue_name         => 'strmadmin.streams_queue',
       capture_name       => 'strm03_capture',
       rule_set_name      => 'strmadmin.strm01_rule_set',
       start_scn          => 223525,
       source_database    => NULL,
       first_scn          => NULL);
   END;
   /
   ```

   Running this procedure performs the following actions:

   - Creates a capture process named `strm03_capture`. A capture process with the same name must not exist.
   - Associates the capture process with the existing queue `strmadmin.streams_queue`.
   - Associates the capture process with the existing rule set `strmadmin.strm01_rule_set`. This rule set is the positive rule set for the capture process.
   - Specifies `223525` as the start SCN for the capture process. The new capture process uses the same LogMiner data dictionary as one of the existing capture processes. Oracle Streams automatically chooses which LogMiner data dictionary to share with the new capture process. Because the `first_scn` parameter was set to `NULL`, the first SCN for the new capture process is the

same as the first SCN of the existing capture process whose LogMiner data dictionary was shared. In this example, the existing capture process is `capture_simp`.

- Creates a capture process that captures local changes to the source database because the `source_database` parameter is set to `NULL`. For a local capture process, you can also specify the global name of the local database for this parameter.

Note: If no local capture process exists when the procedure in this example is run, then the `DBMS_CAPTURE_ADM.BUILD` procedure is run automatically during capture process creation to extract the data dictionary into the redo log. The first time the new capture process is started, it uses this redo data to create a LogMiner data dictionary. In this case, a specified `start_scn` parameter value must be equal to or higher than the current database SCN.

---

5. If necessary, complete the steps described in "After Configuring a Capture Process" on page 5-17.

See Also:

- *Oracle Streams Concepts and Administration*
- *Oracle Database PL/SQL Packages and Types Reference* for more information about setting the `first_scn` and `start_scn` parameters in the `CREATE_CAPTURE` procedure

Configuring a Downstream Capture Process

This section describes configuring a real-time or archived-log downstream capture process.

This section contains these topics:

- Configuring a Real-Time Downstream Capture Process
- Configuring an Archived-Log Downstream Capture Process

Configuring a Real-Time Downstream Capture Process

To create a capture process that performs downstream capture, you must use the `CREATE_CAPTURE` procedure. The example in this section describes creating a real-time downstream capture process that uses a database link to the source database. However, a real-time downstream capture process might not use a database link.

This example assumes the following:

- The source database is `dbs1.example.com` and the downstream database is `dbs2.example.com`.
- The capture process that will be created at `dbs2.example.com` uses the `strmadmin.streams_queue`.
- The capture process will capture DML changes to the `hr.departments` table.

This section contains an example that runs the `CREATE_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package to create a real-time downstream capture process at the `dbs2.example.com` downstream database that captures changes made to the `dbs1.example.com` source database. The capture process in this example uses a

database link to dbs1.example.com for administrative purposes. The name of the database link must match the global name of the source database.

Note: You can configure multiple real-time downstream capture processes that captures changes from the same source database, but you cannot configure real-time downstream capture for multiple source databases at one downstream database.

See Also: *Oracle Streams Concepts and Administration* for conceptual information about real-time downstream capture

Complete the following steps:

1. Complete the tasks in "Preparing to Configure a Capture Process" on page 5-2. Ensure that you complete the tasks "Configuring Log File Transfer to a Downstream Capture Database" on page 1-33 and "Adding Standby Redo Logs for Real-Time Downstream Capture" on page 1-36.

2. In SQL*Plus, connect to the downstream database dbs2.example.com as the Oracle Streams administrator.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.

3. Create the database link from dbs2.example.com to dbs1.example.com. For example, if the user strmadmin is the Oracle Streams administrator on both databases, then create the following database link:

   ```
   CREATE DATABASE LINK dbs1.example.com CONNECT TO strmadmin
      IDENTIFIED BY password
      USING 'dbs1.example.com';
   ```

   This example assumes that an Oracle Streams administrator exists at the source database dbs1.example.com. If no Oracle Streams administrator exists at the source database, then the Oracle Streams administrator at the downstream database should connect to a user who allows remote access by an Oracle Streams administrator. You can enable remote access for a user by specifying the user as the grantee when you run the GRANT_REMOTE_ADMIN_ACCESS procedure in the DBMS_STREAMS_AUTH package at the source database.

4. Run the CREATE_CAPTURE procedure to create the capture process:

   ```
   BEGIN
     DBMS_CAPTURE_ADM.CREATE_CAPTURE(
       queue_name         => 'strmadmin.streams_queue',
       capture_name       => 'real_time_capture',
       rule_set_name      => NULL,
       start_scn          => NULL,
       source_database    => 'dbs1.example.com',
       use_database_link  => TRUE,
       first_scn          => NULL,
       logfile_assignment => 'implicit');
   END;
   /
   ```

   Running this procedure performs the following actions:

- Creates a capture process named `real_time_capture` at the downstream database `dbs2.example.com`. A capture process with the same name must not exist.
- Associates the capture process with an existing queue on `dbs2.example.com` named `streams_queue` and owned by `strmadmin`.
- Specifies that the source database of the changes that the capture process will capture is `dbs1.example.com`.
- Specifies that the capture process uses a database link with the same name as the source database global name to perform administrative actions at the source database.
- Specifies that the capture process accepts redo data implicitly from `dbs1.example.com`. Therefore, the capture process scans the standby redo log at `dbs2.example.com` for changes that it must capture. If the capture process falls behind, then it scans the archived redo log files written from the standby redo log.

This step does not associate the capture process `real_time_capture` with any rule set. A rule set will be created and associated with the capture process in a later step.

If no other capture process at `dbs2.example.com` is capturing changes from the `dbs1.example.com` source database, then the `DBMS_CAPTURE_ADM.BUILD` procedure is run automatically at `dbs1.example.com` using the database link. Running this procedure extracts the data dictionary at `dbs1.example.com` to the redo log, and a LogMiner data dictionary for `dbs1.example.com` is created at `dbs2.example.com` when the capture process `real_time_capture` is started for the first time at `dbs2.example.com`.

If multiple capture processes at `dbs2.example.com` are capturing changes from the `dbs1.example.com` source database, then the new capture process `real_time_capture` uses the same LogMiner data dictionary for `dbs1.example.com` as one of the existing archived-log capture process. Oracle Streams automatically chooses which LogMiner data dictionary to share with the new capture process.

Note: During the creation of a downstream capture process, if the `first_scn` parameter is set to `NULL` in the `CREATE_CAPTURE` procedure, then the `use_database_link` parameter must be set to `TRUE`. Otherwise, an error is raised.

See Also: *Oracle Streams Concepts and Administration* for more information about SCN values related to a capture process

5. Set the `downstream_real_time_mine` capture process parameter to `Y`:

```
BEGIN
  DBMS_CAPTURE_ADM.SET_PARAMETER(
    capture_name => 'real_time_capture',
    parameter    => 'downstream_real_time_mine',
    value        => 'Y');
END;
/
```

6. Create the positive rule set for the capture process and add a rule to it:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name          =>  'hr.departments',
    streams_type        =>  'capture',
    streams_name        =>  'real_time_capture',
    queue_name          =>  'strmadmin.streams_queue',
    include_dml         =>  TRUE,
    include_ddl         =>  FALSE,
    include_tagged_lcr  =>  FALSE,
    source_database     =>  'dbs1.example.com',
    inclusion_rule      =>  TRUE);
END;
/
```

Running this procedure performs the following actions:

- Creates a rule set at `dbs2.example.com` for capture process `real_time_capture`. The rule set has a system-generated name. The rule set is the positive rule set for the capture process because the `inclusion_rule` parameter is set to `TRUE`.
- Creates a rule that captures data manipulation language (DML) changes to the `hr.departments` table, and adds the rule to the positive rule set for the capture process. The rule has a system-generated name. The rule is added to the positive rule set for the capture process because the `inclusion_rule` parameter is set to `TRUE`.
- Prepares the `hr.departments` table at `dbs1.example.com` for instantiation using the database link created in Step 3.
- Enables supplemental logging for any primary key, unique key, bitmap index, and foreign key columns in the table at `dbs1.example.com`.

7. Connect to the source database `dbs1.example.com` as an administrative user with the necessary privileges to switch log files.
8. Archive the current log file at the source database:

   ```
   ALTER SYSTEM ARCHIVE LOG CURRENT;
   ```

   Archiving the current log file at the source database starts real time mining of the source database redo log.

   If the capture process appears to be waiting for redo data for an inordinately long time, then check the alert log for errors. See *Oracle Streams Concepts and Administration* for more information.
9. If necessary, complete the steps described in "After Configuring a Capture Process" on page 5-17.

Configuring an Archived-Log Downstream Capture Process

This section describes configuring an archived-log downstream capture process that either assigns log files implicitly or explicitly.

This section contains these topics:

- Configuring an Archived-Log Downstream Capture Process that Assigns Logs Implicitly
- Configuring an Archived-Log Downstream Capture Process that Assigns Logs Explicitly

Configuring an Archived-Log Downstream Capture Process that Assigns Logs Implicitly To create a capture process that performs downstream capture, you must use the `CREATE_CAPTURE` procedure. The example in this section describes creating an archived-log downstream capture process that uses a database link to the source database for administrative purposes. The name of the database link must match the global name of the source database.

This example assumes the following:

- The source database is `dbs1.example.com` and the downstream database is `dbs2.example.com`.
- The capture process that will be created at `dbs2.example.com` uses the `streams_queue` owned by `strmadmin`.
- The capture process will capture data manipulation language (DML) changes made to the `hr.departments` table at `dbs1.example.com`.
- The capture process assigns log files implicitly. That is, the downstream capture process automatically scans all redo log files added by redo transport services or added manually from the source database to the downstream database.

Complete the following steps:

1. Complete the tasks in "Preparing to Configure a Capture Process" on page 5-2. Ensure that you complete the task "Configuring Log File Transfer to a Downstream Capture Database" on page 1-33.
2. In SQL*Plus, connect to the downstream database `dbs2.example.com` as the Oracle Streams administrator.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.
3. Create the database link from `dbs2.example.com` to `dbs1.example.com`. For example, if the user `strmadmin` is the Oracle Streams administrator on both databases, then create the following database link:

   ```
   CREATE DATABASE LINK dbs1.example.com CONNECT TO strmadmin
      IDENTIFIED BY password
      USING 'dbs1.example.com';
   ```

   This example assumes that an Oracle Streams administrator exists at the source database `dbs1.example.com`. If no Oracle Streams administrator exists at the source database, then the Oracle Streams administrator at the downstream database should connect to a user who allows remote access by an Oracle Streams administrator. You can enable remote access for a user by specifying the user as the grantee when you run the `GRANT_REMOTE_ADMIN_ACCESS` procedure in the `DBMS_STREAMS_AUTH` package at the source database.
4. Run the `CREATE_CAPTURE` procedure to create the capture process:

   ```
   BEGIN
     DBMS_CAPTURE_ADM.CREATE_CAPTURE(
       queue_name         => 'strmadmin.streams_queue',
       capture_name       => 'strm04_capture',
       rule_set_name      => NULL,
       start_scn          => NULL,
       source_database    => 'dbs1.example.com',
       use_database_link  => TRUE,
       first_scn          => NULL,
       logfile_assignment => 'implicit');
   END;
   ```

```
/
```

Running this procedure performs the following actions:

- Creates a capture process named `strm04_capture` at the downstream database `dbs2.example.com`. A capture process with the same name must not exist.
- Associates the capture process with an existing queue on `dbs2.example.com` named `streams_queue` and owned by `strmadmin`.
- Specifies that the source database of the changes that the capture process will capture is `dbs1.example.com`.
- Specifies that the capture process accepts new redo log files implicitly from `dbs1.example.com`. Therefore, the capture process scans any new log files copied from `dbs1.example.com` to `dbs2.example.com` for changes that it must capture.

This step does not associate the capture process `strm04_capture` with any rule set. A rule set will be created and associated with the capture process in the next step.

If no other capture process at `dbs2.example.com` is capturing changes from the `dbs1.example.com` source database, then the `DBMS_CAPTURE_ADM.BUILD` procedure is run automatically at `dbs1.example.com` using the database link. Running this procedure extracts the data dictionary at `dbs1.example.com` to the redo log, and a LogMiner data dictionary for `dbs1.example.com` is created at `dbs2.example.com` when the capture process is started for the first time at `dbs2.example.com`.

If multiple capture processes at `dbs2.example.com` are capturing changes from the `dbs1.example.com` source database, then the new capture process uses the same LogMiner data dictionary for `dbs1.example.com` as one of the existing capture process. Oracle Streams automatically chooses which LogMiner data dictionary to share with the new capture process.

Note: During the creation of a downstream capture process, if the `first_scn` parameter is set to `NULL` in the `CREATE_CAPTURE` procedure, then the `use_database_link` parameter must be set to `TRUE`. Otherwise, an error is raised.

See Also:

- *Oracle Streams Concepts and Administration*
- *Oracle Database SQL Language Reference* for more information about the `ALTER DATABASE` statement
- *Oracle Data Guard Concepts and Administration* for more information registering redo log files

5. Create the positive rule set for the capture process and add a rule to it:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name          =>  'hr.departments',
    streams_type        =>  'capture',
    streams_name        =>  'strm04_capture',
    queue_name          =>  'strmadmin.streams_queue',
```

```
    include_dml         =>  TRUE,
    include_ddl         =>  FALSE,
    include_tagged_lcr  =>  FALSE,
    source_database     =>  'dbs1.example.com',
    inclusion_rule      =>  TRUE);
END;
/
```

Running this procedure performs the following actions:

- Creates a rule set at `dbs2.example.com` for capture process `strm04_capture`. The rule set has a system-generated name. The rule set is a positive rule set for the capture process because the `inclusion_rule` parameter is set to `TRUE`.
- Creates a rule that captures DML changes to the `hr.departments` table, and adds the rule to the rule set for the capture process. The rule has a system-generated name. The rule is added to the positive rule set for the capture process because the `inclusion_rule` parameter is set to `TRUE`.

6. If necessary, complete the steps described in "After Configuring a Capture Process" on page 5-17.

Configuring an Archived-Log Downstream Capture Process that Assigns Logs Explicitly To create a capture process that performs downstream capture, you must use the `CREATE_CAPTURE` procedure. This section describes creating an archived-log downstream capture process that assigns redo log files explicitly. That is, you must use the `DBMS_FILE_TRANSFER` package, FTP, or some other method to transfer redo log files from the source database to the downstream database, and then you must register these redo log files with the downstream capture process manually.

In this example, assume the following:

- The source database is `dbs1.example.com` and the downstream database is `dbs2.example.com`.
- The capture process that will be created at `dbs2.example.com` uses the `streams_queue` owned by `strmadmin`.
- The capture process will capture data manipulation language (DML) changes made to the `hr.departments` table at `dbs1.example.com`.
- The capture process does not use a database link to the source database for administrative actions.

Complete the following steps:

1. Complete the tasks in "Preparing to Configure a Capture Process" on page 5-2. Because in this example you are transferring and registering archived redo log files explicitly at the downstream database, you do not need to complete the task "Configuring Log File Transfer to a Downstream Capture Database" on page 1-33.
2. In SQL*Plus, connect to the source database `dbs1.example.com` as the Oracle Streams administrator.

   If you do not use a database link from the downstream database to the source database, then an Oracle Streams administrator must exist at the source database.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.
3. If there is no capture process at `dbs2.example.com` that captures changes from `dbs1.example.com`, then perform a build of the `dbs1.example.com` data

dictionary in the redo log. This step is optional if a capture process at `dbs2.example.com` is already configured to capture changes from the `dbs1.example.com` source database.

```
SET SERVEROUTPUT ON
DECLARE
  scn  NUMBER;
BEGIN
  DBMS_CAPTURE_ADM.BUILD(
    first_scn => scn);
  DBMS_OUTPUT.PUT_LINE('First SCN Value = ' || scn);
END;
/
First SCN Value = 409391
```

This procedure displays the valid first SCN value for the capture process that will be created at `dbs2.example.com`. Make a note of the SCN value returned because you will use it when you create the capture process at `dbs2.example.com`.

If you run this procedure to build the data dictionary in the redo log, then when the capture process is first started at `dbs2.example.com`, it will create a LogMiner data dictionary using the data dictionary information in the redo log.

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

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

Primary key supplemental logging is required for the `hr.departments` table because this example creates a capture processes that captures changes to this table. Specifying `keys` for the `supplemental_logging` parameter in the `PREPARE_TABLE_INSTANTIATION` procedure enables supplemental logging for any primary key, unique key, bitmap index, and foreign key columns in the table.

5. Determine the current SCN of the source database:

```
SET SERVEROUTPUT ON SIZE 1000000

DECLARE
  iscn  NUMBER;         -- Variable to hold instantiation SCN value
BEGIN
  iscn := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER();
  DBMS_OUTPUT.PUT_LINE('Current SCN: ' || iscn);
END;
/
```

You can use the returned SCN as the instantiation SCN for destination databases that will apply changes to the `hr.departments` table that were captured by the capture process being created. In this example, assume the returned SCN is `1001656`.

6. Connect to the downstream database `dbs2.example.com` as the Oracle Streams administrator.

7. Run the `CREATE_CAPTURE` procedure to create the capture process and specify the value obtained in Step 3 for the `first_scn` parameter:

```
BEGIN
  DBMS_CAPTURE_ADM.CREATE_CAPTURE(
    queue_name         => 'strmadmin.streams_queue',
    capture_name       => 'strm05_capture',
    rule_set_name      => NULL,
    start_scn          => NULL,
    source_database    => 'dbs1.example.com',
    use_database_link  => FALSE,
    first_scn          => 409391, -- Use value from Step 3
    logfile_assignment => 'explicit');
END;
/
```

Running this procedure performs the following actions:

- Creates a capture process named `strm05_capture` at the downstream database `dbs2.example.com`. A capture process with the same name must not exist.
- Associates the capture process with an existing queue on `dbs2.example.com` named `streams_queue` and owned by `strmadmin`.
- Specifies that the source database of the changes that the capture process will capture is `dbs1.example.com`.
- Specifies that the first SCN for the capture process is `409391`. This value was obtained in Step 3. The first SCN is the lowest SCN for which a capture process can capture changes. Because a first SCN is specified, the capture process creates a new LogMiner data dictionary when it is first started, regardless of whether there are existing LogMiner data dictionaries for the same source database.
- Specifies that new redo log files from `dbs1.example.com` must be assigned to the capture process explicitly. After a redo log file has been transferred to the computer running the downstream database, you assign the log file to the capture process explicitly using the following statement:

  ```
  ALTER DATABASE REGISTER LOGICAL LOGFILE file_name FOR capture_process;
  ```

  Here, *`file_name`* is the name of the redo log file and *`capture_process`* is the name of the capture process that will use the redo log file at the downstream database. You must add redo log files manually if the `logfile_assignment` parameter is set to `explicit`.

This step does not associate the capture process `strm05_capture` with any rule set. A rule set will be created and associated with the capture process in the next step.

See Also:

- *Oracle Streams Concepts and Administration* for more information about SCN values related to a capture process
- *Oracle Database SQL Language Reference* for more information about the `ALTER DATABASE` statement
- *Oracle Data Guard Concepts and Administration* for more information registering redo log files

8. Create the positive rule set for the capture process and add a rule to it:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name          =>  'hr.departments',
    streams_type        =>  'capture',
    streams_name        =>  'strm05_capture',
    queue_name          =>  'strmadmin.streams_queue',
    include_dml         =>  TRUE,
    include_ddl         =>  FALSE,
    include_tagged_lcr  =>  FALSE,
    source_database     =>  'dbs1.example.com',
    inclusion_rule      =>  TRUE);
END;
/
```

Running this procedure performs the following actions:

- Creates a rule set at `dbs2.example.com` for capture process `strm05_capture`. The rule set has a system-generated name. The rule set is a positive rule set for the capture process because the `inclusion_rule` parameter is set to `TRUE`.
- Creates a rule that captures DML changes to the `hr.departments` table, and adds the rule to the rule set for the capture process. The rule has a system-generated name. The rule is added to the positive rule set for the capture process because the `inclusion_rule` parameter is set to `TRUE`.

9. After the redo log file at the source database `dbs1.example.com` that contains the first SCN for the downstream capture process is archived, transfer the archived redo log file to the computer running the downstream database. The `BUILD` procedure in Step 3 determined the first SCN for the downstream capture process. If the redo log file is not yet archived, then you can run the `ALTER SYSTEM SWITCH LOGFILE` statement on the database to archive it.

   You can run the following query at `dbs1.example.com` to identify the archived redo log file that contains the first SCN for the downstream capture process:

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

   SELECT NAME, FIRST_CHANGE# FROM V$ARCHIVED_LOG
     WHERE FIRST_CHANGE# IS NOT NULL AND DICTIONARY_BEGIN = 'YES';
   ```

   Transfer the archived redo log file with a `FIRST_CHANGE#` that matches the first SCN returned in Step 3 to the computer running the downstream capture process.

10. Connect to the downstream database `dbs2.example.com` as an administrative user.

11. Assign the transferred redo log file to the capture process. For example, if the redo log file is `/oracle/logs_from_dbs1/1_10_486574859.dbf`, then issue the following statement:

    ```
    ALTER DATABASE REGISTER LOGICAL LOGFILE
       '/oracle/logs_from_dbs1/1_10_486574859.dbf' FOR 'strm05_capture';
    ```

12. If necessary, complete the steps described in "After Configuring a Capture Process" on page 5-17.

After Configuring a Capture Process

If you plan to configure propagations and apply processes that process logical change records (LCRs) captured by the new capture process, then perform the configuration in the following order:

1. Create all of the queues that will be required propagations and apply processes in the replication environment. See "Creating an ANYDATA Queue" on page 6-1.
2. Create all of the propagations that will propagate LCRs captured by the new capture process. See "Creating Oracle Streams Propagations Between ANYDATA Queues" on page 6-3.
3. Create all of the apply processes that will dequeue and process LCRs captured by the new capture process. See Chapter 7, "Configuring Implicit Apply". Configure each apply process to apply captured LCRs.
4. Instantiate the tables for which the new capture process captures changes at all destination databases. See Chapter 8, "Instantiation and Oracle Streams Replication" for detailed information about instantiation.
5. Use the `START_APPLY` procedure in the `DBMS_APPLY_ADM` package to start the apply processes that will process LCRs captured by the new capture process, or see *Oracle Database 2 Day + Data Replication and Integration Guide* for instructions about starting an apply process using Oracle Enterprise Manager.
6. Use the `START_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package to start the new capture process, or see *Oracle Database 2 Day + Data Replication and Integration Guide* for instructions about starting a capture process using Oracle Enterprise Manager.

Note: Other configuration steps might be required for your Oracle Streams environment. For example, some Oracle Streams environments include transformations, apply handlers, and conflict resolution.

Configuring Synchronous Capture

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

- `DBMS_STREAMS_ADM.ADD_TABLE_RULES`
- `DBMS_STREAMS_ADM.ADD_SUBSET_RULES`
- `DBMS_CAPTURE_ADM.CREATE_SYNC_CAPTURE`

Both of the procedures in the `DBMS_STREAMS_ADM` package create a synchronous capture with the specified name if it does not already exist, create a positive rule set for the synchronous capture if it does not exist, and can add table rules or subset rules to the rule set.

The `CREATE_SYNC_CAPTURE` procedure creates a synchronous capture, but does not create a rule set or rules for the synchronous capture. However, the `CREATE_SYNC_CAPTURE` procedure enables you to specify an existing rule set to associate with the synchronous capture, and it enables you to specify a capture user other than the default capture user.

The following sections describe configuring a synchronous capture:

- Preparing to Configure a Synchronous Capture
- Configuring a Synchronous Capture Using the DBMS_STREAMS_ADM Package

- Configuring a Synchronous Capture Using the DBMS_CAPTURE_ADM Package
- After Configuring a Synchronous Capture

See Also:

- *Oracle Streams Concepts and Administration*
- *Oracle Database 2 Day + Data Replication and Integration Guide* for an example that configures a replication environment that uses synchronous capture

Preparing to Configure a Synchronous Capture

The following tasks must be completed before you configure a synchronous capture:

- Complete the following tasks in "Tasks to Complete Before Configuring Oracle Streams Replication" on page 1-13.
  - "Configuring an Oracle Streams Administrator on All Databases" on page 1-13
  - "Configuring Network Connectivity and Database Links" on page 1-17
  - "Setting Initialization Parameters Relevant to Oracle Streams" on page 1-19
  - "Configuring the Oracle Streams Pool" on page 1-25
- Create `ANYDATA` queues to associate with the synchronous capture, if they do not exist. See "Creating an ANYDATA Queue" on page 6-1 for instructions. The queue must be a commit-time queue. The examples in this chapter assume that the queue used by synchronous capture is `strmadmin.streams_queue`. Create the queue in the same database that will run the synchronous capture.
- Create `ANYDATA` queues to associate with the propagations that will propagate logical change records (LCRs) captured by the synchronous capture and apply processes that will dequeue and process LCRs captured by the synchronous capture, if they do not exist. See "Creating an ANYDATA Queue" on page 6-1 for instructions.
- Create all of the propagations that will propagate LCRs captured by the new synchronous capture. See "Creating Oracle Streams Propagations Between ANYDATA Queues" on page 6-3.
- Create all of the apply processes that will dequeue and process LCRs captured by the new synchronous capture. See "Creating an Apply Process for Persistent LCRs with DBMS_APPLY_ADM" on page 7-6. Configure each apply process to apply persistent LCRs by setting the `apply_captured` parameter to `FALSE` in the `DBMS_APPLY_ADM.CREATE_APPLY` procedure. Do not start the apply process until after the instantiation performed in "After Configuring a Synchronous Capture" on page 5-22 is complete.
- Ensure that the Oracle Streams administrator is granted `DBA` role. The Oracle Streams administrator must be granted `DBA` role to create a synchronous capture.

Configuring a Synchronous Capture Using the DBMS_STREAMS_ADM Package

When you run the `ADD_TABLE_RULES` or `ADD_SUBSET_RULES` procedure to create a synchronous capture, set the `streams_type` parameter in these procedures to `sync_capture`. A rule created by the `ADD_TABLE_RULES` procedure instructs the synchronous capture to capture all data manipulation language (DML) changes to the table. A rule created by the `ADD_SUBSET_RULES` procedure instructs the synchronous capture to capture a subset of the DML changes to the table.

This example assumes the following:

- The source database is `dbs1.example.com`.
- The synchronous capture that will be created uses the `strmadmin.streams_queue` queue.
- The synchronous capture that will be created captures the results of DML changes made to the `hr.departments` table.
- The capture user for the synchronous capture that will be created is the Oracle Streams administrator `strmadmin`.

Complete the following steps to create a synchronous capture using the `DBMS_STREAMS_ADM` package:

1. Complete the tasks in "Preparing to Configure a Synchronous Capture" on page 5-18.
2. In SQL*Plus, connect to the `dbs1.example.com` database as the Oracle Streams administrator.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.
3. Run the `ADD_TABLE_RULES` or `ADD_SUBSET_RULES` procedure to create a synchronous capture. For example:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name   => 'hr.departments',
    streams_type => 'sync_capture',
    streams_name => 'sync_capture',
    queue_name   => 'strmadmin.streams_queue');
END;
/
```

   This procedure performs the following actions:

   - Creates a synchronous capture named `sync_capture` at the source database.
   - Enables the synchronous capture. A synchronous capture cannot be disabled.
   - Associates the synchronous capture with the existing `strmadmin.streams_queue` queue.
   - Creates a positive rule set for the synchronous capture. The rule set has a system-generated name.
   - Creates a rule that captures DML changes to the `hr.departments` table, and adds the rule to the positive rule set for the synchronous capture. The rule has a system-generated name.
   - Configures the user who runs the procedure as the capture user for the synchronous capture. In this case, this user is `strmadmin`.
   - Prepares the specified table for instantiation by running the `DBMS_CAPTURE_ADM.PREPARE_SYNC_INSTANTIATION` function for the table automatically.

> **Note:** When the ADD_TABLE_RULES or the ADD_SUBSET_RULES procedure adds rules to a synchronous capture rule set, the procedure must obtain an exclusive lock on the specified table. If there are outstanding transactions on the specified table, then the procedure waits until it can obtain a lock.

4. If necessary, complete the steps described in "After Configuring a Synchronous Capture" on page 5-22.

Configuring a Synchronous Capture Using the DBMS_CAPTURE_ADM Package

This section contains an example that runs procedures in the DBMS_CAPTURE_ADM package and DBMS_STREAMS_ADM package to configure a synchronous capture.

This example assumes the following:

- The source database is dbs1.example.com.
- The synchronous capture that will be created uses the strmadmin.streams_queue queue.
- The synchronous capture that will be created uses an existing rule set named sync01_rule_set in the strmadmin schema.
- The synchronous capture that will be created captures the results of a subset of the DML changes made to the hr.departments table.
- The capture user for the synchronous capture that will be created is hr. The hr user must have privileges to enqueue into the streams_queue.

Complete the following steps to create a synchronous capture using the DBMS_CAPTURE_ADM package:

1. Complete the tasks in "Preparing to Configure a Synchronous Capture" on page 5-18.
2. In SQL*Plus, connect to the dbs1.example.com database as the Oracle Streams administrator.

   See *Oracle Database Administrator's Guide* for information about connecting to a database in SQL*Plus.
3. Create the rule set that will be used by the synchronous capture if it does not exist. In this example, assume that the rule set is strmadmin.sync01_rule_set. See *Oracle Streams Concepts and Administration* for instructions.
4. Run the CREATE_SYNC_CAPTURE procedure to create a synchronous capture. For example:

```
BEGIN
  DBMS_CAPTURE_ADM.CREATE_SYNC_CAPTURE(
    queue_name    => 'strmadmin.streams_queue',
    capture_name  => 'sync01_capture',
    rule_set_name => 'strmadmin.sync01_rule_set',
    capture_user  => 'hr');
END;
/
```

   Running this procedure performs the following actions:

- Creates a synchronous capture named `sync01_capture`. A synchronous capture with the same name must not exist.
- Enables the synchronous capture. A synchronous capture cannot be disabled.
- Associates the synchronous capture with the existing queue `strmadmin.streams_queue`.
- Associates the synchronous capture with the existing rule set `strmadmin.sync01_rule_set`.
- Configures `hr` as the capture user for the synchronous capture.

5. Run the `ADD_TABLE_RULES` or `ADD_SUBSET_RULES` procedure to add a rule to the synchronous capture rule set. For example, run the `ADD_SUBSET_RULES` procedure to instruct the synchronous capture to capture a subset of the DML changes to the `hr.departments` table:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_SUBSET_RULES(
    table_name          => 'hr.departments',
    dml_condition       => 'department_id=1700',
    streams_type        => 'sync_capture',
    streams_name        => 'sync01_capture',
    queue_name          => 'strmadmin.streams_queue',
    include_tagged_lcr  =>  FALSE);
END;
/
```

Running this procedure performs the following actions:

- Adds subset rules to the rule set for the synchronous capture named `sync01_capture` at the source database `dbs1.example.com`. The subset rules instruct the synchronous capture to capture changes to rows with `department_id` equal to `1700`. The synchronous capture does not capture changes to other rows in the table.
- Prepares the `hr.departments` table for instantiation by running the `DBMS_CAPTURE_ADM.PREPARE_SYNC_INSTANTIATION` function for the table automatically.
- Specifies that the synchronous capture captures a change only if the session that makes 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 synchronous capture.

Note: When the `CREATE_SYNC_CAPTURE` procedure creates a synchronous capture, the procedure must obtain an exclusive lock on each table for which it will capture changes. The rules in the specified rule set for the synchronous capture determine these tables. Similarly, when the `ADD_TABLE_RULES` or the `ADD_SUBSET_RULES` procedure adds rules to a synchronous capture rule set, the procedure must obtain an exclusive lock on the specified table. In these cases, if there are outstanding transactions on a table for which the synchronous capture will capture changes, then the procedure waits until it can obtain a lock.

---

6. If necessary, complete the steps described in "After Configuring a Synchronous Capture" on page 5-22.

After Configuring a Synchronous Capture

If you configured propagations and apply processes that process logical change records (LCRs captured) by the new synchronous capture, then complete the following steps:

1. Instantiate the tables for which the new synchronous capture captures changes at all destination databases. See Chapter 8, "Instantiation and Oracle Streams Replication" for detailed information about instantiation.
2. Use the `START_APPLY` procedure in the `DBMS_APPLY_ADM` package to start the apply processes that will process LCRs captured by the new synchronous capture, or see *Oracle Database 2 Day + Data Replication and Integration Guide* for instructions about starting an apply process using Oracle Enterprise Manager.

Note: Other configuration steps might be required for your Oracle Streams environment. For example, some Oracle Streams environments include transformations, apply handlers, and conflict resolution.

6

Configuring Queues and Propagations

The following topics describe configuring queues and propagations:

- Creating an ANYDATA Queue
- Creating Oracle Streams Propagations Between ANYDATA Queues

Each task described in this chapter should be completed by an Oracle Streams administrator that has been granted the appropriate privileges, unless specified otherwise.

See Also:

- *Oracle Streams Concepts and Administration* for more information about queues and propagations
- "Configuring an Oracle Streams Administrator on All Databases" on page 1-13

Creating an ANYDATA Queue

A queue stores messages in an Oracle Streams environment. Messages can be enqueued, propagated from one queue to another, and dequeued. An `ANYDATA` queue stores messages whose payloads are of `ANYDATA` type. Therefore, an `ANYDATA` queue can store a message with a payload of nearly any type, if the payload is wrapped in an `ANYDATA` wrapper. Each Oracle Streams capture process, synchronous capture, apply process, and messaging client is associated with one `ANYDATA` queue, and each Oracle Streams propagation is associated with one `ANYDATA` source queue and one `ANYDATA` destination queue.

The easiest way to create an `ANYDATA` queue is to use the `SET_UP_QUEUE` procedure in the `DBMS_STREAMS_ADM` package. This procedure enables you to specify the following settings for the `ANYDATA` queue it creates:

- The queue table for the queue
- A storage clause for the queue table
- The queue name
- A queue user that will be configured as a secure queue user of the queue and granted `ENQUEUE` and `DEQUEUE` privileges on the queue
- A comment for the queue

If the specified queue table does not exist, then it is created. If the specified queue table exists, then the existing queue table is used for the new queue. If you do not specify any queue table when you create the queue, then, by default, `streams_queue_table` is specified.

For example, complete the following steps to create an ANYDATA queue with the SET_UP_QUEUE procedure:

1. Complete the following tasks in "Tasks to Complete Before Configuring Oracle Streams Replication" on page 1-13 you create an ANYDATA queue:
   - "Configuring an Oracle Streams Administrator on All Databases" on page 1-13
   - "Setting Initialization Parameters Relevant to Oracle Streams" on page 1-19
   - "Configuring the Oracle Streams Pool" on page 1-25
2. In SQL*Plus, connect to the database that will contain the queue as the Oracle Streams administrator.

   See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.
3. Run the SET_UP_QUEUE procedure to create the queue:

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

   Running this procedure performs the following actions:
   - Creates a queue table named streams_queue_table in the strmadmin schema. The queue table is created only if it does not already exist. Queues based on the queue table store messages of ANYDATA type. Queue table names can be a maximum of 24 bytes.
   - Creates a queue named streams_queue in the strmadmin schema. The queue is created only if it does not already exist. Queue names can be a maximum of 24 bytes.
   - Specifies that the streams_queue queue is based on the strmadmin.streams_queue_table queue table.
   - Configures the hr user as a secure queue user of the queue, and grants this user ENQUEUE and DEQUEUE privileges on the queue.
   - Starts the queue.

   Default settings are used for the parameters that are not explicitly set in the SET_UP_QUEUE procedure.

When the SET_UP_QUEUE procedure creates a queue table, the following DBMS_AQADM.CREATE_QUEUE_TABLE parameter settings are specified:

- If the database is Oracle Database 10*g* Release 2 or later, the sort_list setting is commit_time. If the database is a release before Oracle Database 10*g* Release 2, the sort_list setting is enq_time.
- The multiple_consumers setting is TRUE.
- The message_grouping setting is transactional.
- The secure setting is TRUE.

The other parameters in the CREATE_QUEUE_TABLE procedure are set to their default values.

You can use the CREATE_QUEUE_TABLE procedure in the DBMS_AQADM package to create a queue table of ANYDATA type with different properties than the default properties specified by the SET_UP_QUEUE procedure in the DBMS_STREAMS_ADM package. After you create the queue table with the CREATE_QUEUE_TABLE procedure, you can create a queue that uses the queue table. To do so, specify the queue table in the queue_table parameter of the SET_UP_QUEUE procedure.

Similarly, you can use the CREATE_QUEUE procedure in the DBMS_AQADM package to create a queue instead of SET_UP_QUEUE. Use CREATE_QUEUE if you require custom settings for the queue. For example, use CREATE_QUEUE to specify a custom retry delay or retention time. If you use CREATE_QUEUE, then you must start the queue manually.

Note:

- You can configure an entire Oracle Streams environment, including queues, using procedures in the DBMS_STREAMS_ADM package or Oracle Enterprise Manager. See Chapter 2, "Simple Oracle Streams Replication Configuration".
- A message cannot be enqueued unless a subscriber who can dequeue the message is configured.

See Also:

- *Oracle Streams Concepts and Administration*
- *Oracle Streams Advanced Queuing User's Guide*
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the SET_UP_QUEUE, CREATE_QUEUE_TABLE, and CREATE_QUEUE procedures

Creating Oracle Streams Propagations Between ANYDATA Queues

A propagation sends messages from an Oracle Streams source queue to an Oracle Streams destination queue. In addition, you can use the features of Oracle Streams Advanced Queuing (AQ) to manage Oracle Streams propagations.

You can use any of the following procedures to create a propagation between two ANYDATA queues:

- DBMS_STREAMS_ADM.ADD_SUBSET_PROPAGATION_RULES
- DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES
- DBMS_STREAMS_ADM.ADD_SCHEMA_PROPAGATION_RULES
- DBMS_STREAMS_ADM.ADD_GLOBAL_PROPAGATION_RULES
- DBMS_PROPAGATION_ADM.CREATE_PROPAGATION

Each of these procedures in the DBMS_STREAMS_ADM package creates a propagation with the specified name if it does not already exist, creates either a positive rule set or negative rule set for the propagation if the propagation does not have such a rule set, and can add table rules, schema rules, or global rules to the rule set.

The CREATE_PROPAGATION procedure creates a propagation, but does not create a rule set or rules for the propagation. However, the CREATE_PROPAGATION procedure enables you to specify an existing rule set to associate with the propagation, either as a

positive or a negative rule set. All propagations are started automatically upon creation.

This section contains the following topics:

- Preparing to Create a Propagation
- Creating a Propagation Using DBMS_STREAMS_ADM
- Creating a Propagation Using DBMS_PROPAGATION_ADM

Note: You can configure an entire Oracle Streams environment, including propagations, using procedures in the `DBMS_STREAMS_ADM` package or Oracle Enterprise Manager. See Chapter 2, "Simple Oracle Streams Replication Configuration".

See Also:

- *Oracle Streams Concepts and Administration*
- *Oracle Streams Advanced Queuing User's Guide* for more information about configuring propagations with the features of Oracle Streams AQ and instructions about configuring propagations between typed queues

Preparing to Create a Propagation

The following tasks must be completed before you create a propagation:

- Complete the following tasks in "Tasks to Complete Before Configuring Oracle Streams Replication" on page 1-13:
  - "Configuring an Oracle Streams Administrator on All Databases" on page 1-13
  - "Configuring Network Connectivity and Database Links" on page 1-17 if the propagation will send messages between databases
  - "Setting Initialization Parameters Relevant to Oracle Streams" on page 1-19
  - "Configuring the Oracle Streams Pool" on page 1-25
- Create a source queue and a destination queue for the propagation, if they do not exist. Both queues must be `ANYDATA` queues. The examples in this chapter assume that the source queue is `strmadmin.strm_a_queue` and that the destination queue is `strmadmin.strm_b_queue`. See "Creating an ANYDATA Queue" on page 6-1 for instructions.

Creating a Propagation Using DBMS_STREAMS_ADM

Complete the following steps to create a propagation using the `ADD_TABLE_PROPAGATION_RULES` procedure in the `DBMS_STREAMS_ADM` package:

1. Complete the tasks in "Preparing to Create a Propagation" on page 6-4.
2. In SQL*Plus, connect to the database that contains the source queue for the propagation as the Oracle Streams administrator.

   See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.
3. Run the `ADD_TABLE_PROPAGATION_RULES` procedure to create the propagation:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_PROPAGATION_RULES(
    table_name              => 'hr.departments',
    streams_name            => 'strm01_propagation',
    source_queue_name       => 'strmadmin.strm_a_queue',
    destination_queue_name  => 'strmadmin.strm_b_queue@dbs2.example.com',
    include_dml             => TRUE,
    include_ddl             => TRUE,
    include_tagged_lcr      => FALSE,
    source_database         => 'dbs1.example.com',
    inclusion_rule          => TRUE,
    queue_to_queue          => TRUE);
END;
/
```

Running this procedure performs the following actions:

- Creates a propagation named `strm01_propagation`. The propagation is created only if it does not already exist.
- Specifies that the propagation propagates logical change records (LCRs) from `strmadmin.strm_a_queue` in the current database to `strmadmin.strm_b_queue` in the `dbs2.example.com` database. These queues must exist.
- Specifies that the propagation uses the `dbs2.example.com` database link to propagate the LCRs, because the `destination_queue_name` parameter contains `@dbs2.example.com`. This database link must exist.
- Creates a positive rule set and associates it with the propagation because the `inclusion_rule` parameter is set to `TRUE`. The rule set uses the evaluation context `SYS.STREAMS$_EVALUATION_CONTEXT`. The rule set name is system generated.
- Creates two rules. One rule evaluates to `TRUE` for row LCRs that contain the results of data manipulation language (DML) changes to the `hr.departments` table. The other rule evaluates to `TRUE` for DDL LCRs that contain data definition language (DDL) changes to the `hr.departments` table. The rule names are system generated.
- Adds the two rules to the positive rule set associated with the propagation. The rules are added to the positive rule setbecause the `inclusion_rule` parameter is set to `TRUE`.
- Specifies that the propagation propagates an LCR only if it has a `NULL` tag, because the `include_tagged_lcr` parameter is set to `FALSE`. This behavior is accomplished through the system-created rules for the propagation.
- Specifies that the source database for the LCRs being propagated is `dbs1.example.com`, which might or might not be the current database. This propagation does not propagate LCRs in the source queue that have a different source database.
- Creates a propagation job for the queue-to-queue propagation.

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

See Also:

- *Oracle Streams Concepts and Administration* for more information about queues, propagations, and rules
- Chapter 10, "Oracle Streams Tags"

Creating a Propagation Using DBMS_PROPAGATION_ADM

Complete the following steps to create a propagation using the `CREATE_PROPAGATION` procedure in the `DBMS_PROPAGATION_ADM` package:

1. Complete the tasks in "Preparing to Create a Propagation" on page 6-4.
2. In SQL*Plus, connect to the database that contains the source queue for the propagation as the Oracle Streams administrator.

   See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.
3. Create the rule set that will be used by the propagation if it does not exist. In this example, assume that the rule set is `strmadmin.strm01_rule_set`. Optionally, you can also add rules to the rule set. See *Oracle Streams Concepts and Administration* for instructions.
4. Run the `CREATE_PROPAGATION` procedure to create the propagation:

```
BEGIN
  DBMS_PROPAGATION_ADM.CREATE_PROPAGATION(
    propagation_name   => 'strm02_propagation',
    source_queue       => 'strmadmin.strm_a_queue',
    destination_queue  => 'strmadmin.strm_b_queue',
    destination_dblink => 'dbs2.example.com',
    rule_set_name      => 'strmadmin.strm01_rule_set',
    queue_to_queue     => TRUE);
END;
/
```

   Running this procedure performs the following actions:

   - Creates a propagation named `strm02_propagation`. A propagation with the same name must not exist.
   - Specifies that the propagation propagates messages from `strmadmin.strm_a_queue` in the current database to `strmadmin.strm_b_queue` in the `dbs2.example.com` database. These queues must exist. Depending on the rules in the rule sets for the propagation, the propagated messages can be LCRs or user messages, or both.
   - Specifies that the propagation uses the `dbs2.example.com` database link to propagate the messages. This database link must exist.
   - Associates the propagation with the rule set named `strmadmin.strm01_rule_set`. This rule set must exist. This rule set is the positive rule set for the propagation.
   - Creates a propagation job for the queue-to-queue propagation.

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

7

Configuring Implicit Apply

In a replication environment, Oracle Streams apply process dequeues logical change records (LCRs) from a specific queue and either applies each one directly or passes it as a parameter to a user-defined procedure called an apply handler.

The following topics describe configuring implicit apply:

- Overview of Apply Process Creation
- Creating an Apply Process for Captured LCRs Using DBMS_STREAMS_ADM
- Creating an Apply Process Using DBMS_APPLY_ADM

Each task described in this chapter should be completed by an Oracle Streams administrator that has been granted the appropriate privileges, unless specified otherwise.

See Also:

- *Oracle Streams Concepts and Administration*
- "Configuring an Oracle Streams Administrator on All Databases" on page 1-13

Overview of Apply Process Creation

You can use any of the following procedures to configure an apply process:

- `DBMS_STREAMS_ADM.ADD_TABLE_RULES`
- `DBMS_STREAMS_ADM.ADD_SUBSET_RULES`
- `DBMS_STREAMS_ADM.ADD_SCHEMA_RULES`
- `DBMS_STREAMS_ADM.ADD_GLOBAL_RULES`
- `DBMS_STREAMS_ADM.ADD_MESSAGE_RULE`
- `DBMS_APPLY_ADM.CREATE_APPLY`

Each of the procedures in the `DBMS_STREAMS_ADM` package creates an apply process with the specified name if it does not already exist, creates either a positive rule set or negative rule set for the apply process if the apply process does not have such a rule set, and can add table rules, schema rules, global rules, or a message rule to the rule set.

The `CREATE_APPLY` procedure in the `DBMS_APPLY_ADM` package creates an apply process, but does not create a rule set or rules for the apply process. However, the `CREATE_APPLY` procedure enables you to specify an existing rule set to associate with the apply process, either as a positive or a negative rule set, and several other options,

such as apply handlers, an apply user, an apply tag, and whether to dequeue messages from a buffered queue or a persistent queue.

A single apply process must either dequeue messages from a buffered queue or a persistent queue. Logical change records that were captured by a capture process are called captured LCRs, and they are always in buffered queues. Therefore, if a single apply process applies LCRs that were captured by a capture process, then it cannot apply persistent LCRs or persistent user messages.

Alternatively, LCRs that were captured by a synchronous capture are persistent LCRs, and they are always in persistent queues. Therefore, if a single apply process applies LCRs that were captured by a synchronous capture, then it cannot apply LCRs captured by a capture process. However, a single apply process can apply both persistent LCRs and persistent user messages because both types of messages are staged in a persistent queue.

The examples in this chapter create apply processes that apply captured LCRs, persistent LCRs, and persistent user messages. Before you create an apply process, create an `ANYDATA` queue to associate with the apply process, if one does not exist.

Note:

- You can configure an entire Oracle Streams environment, including apply processes, using procedures in the `DBMS_STREAMS_ADM` package or Oracle Enterprise Manager. See Chapter 2, "Simple Oracle Streams Replication Configuration".
- Depending on the configuration of the apply process you create, supplemental logging might be required at the source database on columns in the tables for which an apply process applies changes. See "Specifying Supplemental Logging" on page 1-28.
- *Oracle Streams Concepts and Administration* for more information about the captured LCRs and persistent LCRs

---

Preparing to Create an Apply Process

The following tasks must be completed before you create an apply:

- Complete the following tasks in "Tasks to Complete Before Configuring Oracle Streams Replication" on page 1-13.
  - "Configuring an Oracle Streams Administrator on All Databases" on page 1-13
  - "Setting Initialization Parameters Relevant to Oracle Streams" on page 1-19
  - "Configuring the Oracle Streams Pool" on page 1-25
- Create an `ANYDATA` queue to associate with the apply process, if one does not exist. See "Creating an ANYDATA Queue" on page 6-1 for instructions. The examples in this chapter assume that the queue used by the apply process is `strmadmin.streams_queue`. Create the queue on the same database that will run the apply process.

Creating an Apply Process for Captured LCRs Using DBMS_STREAMS_ADM

The following example runs the ADD_SCHEMA_RULES procedure in the DBMS_STREAMS_ADM package to create an apply process that applies captured logical change records (LCRs). This apply process can apply LCRs that were captured by a capture process.

Complete the following steps:

1. Complete the tasks in "Preparing to Create an Apply Process" on page 7-2.
2. In SQL*Plus, connect to the database that will run the apply process as the Oracle Streams administrator.

   See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.
3. Create the apply process:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
    schema_name        => 'hr',
    streams_type       => 'apply',
    streams_name       => 'strm01_apply',
    queue_name         => 'strmadmin.streams_queue',
    include_dml        => TRUE,
    include_ddl        => FALSE,
    include_tagged_lcr => FALSE,
    source_database    => 'dbs1.example.com',
    inclusion_rule     => TRUE);
END;
/
```

   Running this procedure performs the following actions:

   - Creates an apply process named strm01_apply that applies captured LCRs to the local database. The apply process is created only if it does not already exist.
   - Associates the apply process with the existing queue strmadmin.streams_queue. This queue must exist.
   - Creates a positive rule set and associates it with the apply process, if the apply process does not have a positive rule set, because the inclusion_rule parameter is set to TRUE. The rule set uses the SYS.STREAMS$_EVALUATION_CONTEXT evaluation context. The rule set name is system generated.
   - Creates one rule that evaluates to TRUE for row LCRs that contain the results of data manipulation language (DML) changes to database objects in the hr schema. The rule name is system generated.
   - Adds the rule to the positive rule set associated with the apply process because the inclusion_rule parameter is set to TRUE.
   - Sets the apply_tag for the apply process to a value that is the hexadecimal equivalent of '00' (double zero). This is the default apply tag value. Redo entries generated by the apply process have a tag with this value.
   - Specifies that the apply process applies a row LCR only if it has a NULL tag, because the include_tagged_lcr parameter is set to FALSE. This behavior is accomplished through the system-created rule for the apply process.

- Specifies that the LCRs applied by the apply process originate at the `dbs1.example.com` source database. The rules in the apply process rule sets determine which LCRs are dequeued by the apply process. If the apply process dequeues an LCR with a source database other than `dbs1.example.com`, then an error is raised.

Creating an Apply Process Using DBMS_APPLY_ADM

This section contains the following examples that create an apply process using the `DBMS_APPLY_ADM` package:

- Creating an Apply Process for Captured LCRs with DBMS_APPLY_ADM
- Creating an Apply Process for Persistent LCRs with DBMS_APPLY_ADM

See Also:

- "Change Apply in an Oracle to Non-Oracle Environment" on page 11-3 for information about configuring an apply process to apply messages to a non-Oracle database using the `apply_database_link` parameter
- *Oracle Streams Concepts and Administration*

Creating an Apply Process for Captured LCRs with DBMS_APPLY_ADM

The following example runs the `CREATE_APPLY` procedure in the `DBMS_APPLY_ADM` package to create an apply process that applies captured logical change records (LCRs). This apply process can apply LCRs that were captured by a capture process.

Complete the following steps:

1. Complete the tasks in "Preparing to Create an Apply Process" on page 7-2.
2. In SQL*Plus, connect to the database that will run the apply process as the Oracle Streams administrator.

   Ensure that the Oracle Streams administrator is granted `DBA` role. `DBA` role is required because this example sets the apply user to a user other than the Oracle Streams administrator.

   See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.
3. Create the rule set that will be used by the apply process if it does not exist. In this example, assume that the rule set is `strmadmin.strm02_rule_set`. Optionally, you can also add rules to the rule set. See *Oracle Streams Concepts and Administration* for instructions.
4. Create any apply handlers that will be used by the apply process if they do not exist. In this example, assume that the DDL handler is the `strmadmin.history_ddl` procedure. An example in the *Oracle Streams Concepts and Administration* creates this procedure.
5. Create the apply process:

```
BEGIN
  DBMS_APPLY_ADM.CREATE_APPLY(
    queue_name             => 'strmadmin.streams_queue',
    apply_name             => 'strm02_apply',
    rule_set_name          => 'strmadmin.strm02_rule_set',
    message_handler        => NULL,
```

```
    ddl_handler            => 'strmadmin.history_ddl',
    apply_user             => 'hr',
    apply_database_link    => NULL,
    apply_tag              => HEXTORAW('5'),
    apply_captured         => TRUE,
    precommit_handler      => NULL,
    negative_rule_set_name => NULL,
    source_database        => 'dbs1.example.com');
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 the queue `strmadmin.streams_queue`. This queue must exist.
- Associates the apply process with the rule set `strmadmin.strm02_rule_set`. This rule set must exist. This rule set is the positive rule set for the apply process.
- Specifies that the apply process does not use a message handler.
- Specifies that the DDL handler is the `history_ddl` PL/SQL procedure in the `strmadmin` schema. This procedure must exist, and the user who runs the `CREATE_APPLY` procedure must have `EXECUTE` privilege on the `history_ddl` PL/SQL procedure.
- Specifies that the user who applies changes is `hr`, and not the user who is running the `CREATE_APPLY` procedure (the Oracle 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'`. See Chapter 10, "Oracle Streams Tags" for more information about tags.
- Specifies that the apply process applies captured LCRs, not persistent LCRs or persistent user messages. Therefore, if an LCR that was constructed by a synchronous capture or a user application, not by a capture process, and is staged in the queue for the apply process, then this apply process does not dequeue the LCR.
- Specifies that the apply process does not use a precommit handler.
- Specifies that the apply process does not use a negative rule set.
- Specifies that the LCRs applied by the apply process originate at the `dbs1.example.com` source database. The rules in the apply process rule sets determine which LCRs are dequeued by the apply process. If the apply process dequeues an LCR with a source database other than `dbs1.example.com`, then an error is raised.

After creating the apply process, run the `ADD_TABLE_RULES` or `ADD_SUBSET_RULES` procedure to add rules to the apply process rule set. These rules direct the apply process to apply LCRs for the specified tables.

> **See Also:** *Oracle Streams Concepts and Administration* for more information about rules

Creating an Apply Process for Persistent LCRs with DBMS_APPLY_ADM

The following example runs the `CREATE_APPLY` procedure in the `DBMS_APPLY_ADM` package to create an apply process that applies persistent logical change records (LCRs). This apply process can apply LCRs that were captured by a synchronous capture or constructed by an application.

Complete the following steps:

1. Complete the tasks in "Preparing to Create an Apply Process" on page 7-2.
2. In SQL*Plus, connect to the database that will run the apply process as the Oracle Streams administrator.

   See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.
3. Create the rule set that will be used by the apply process if it does not exist. In this example, assume that the rule set is `strmadmin.strm03_rule_set`. Optionally, you can also add rules to the rule set. See *Oracle Streams Concepts and Administration* for instructions.
4. Create any apply handlers that will be used by the apply process if they do not exist. The apply process created in this example does not used apply handlers.
5. Create the apply process:

```
BEGIN
  DBMS_APPLY_ADM.CREATE_APPLY(
    queue_name             => 'strmadmin.streams_queue',
    apply_name             => 'strm03_apply',
    rule_set_name          => 'strmadmin.strm03_rule_set',
    message_handler        => NULL,
    ddl_handler            => NULL,
    apply_user             => NULL,
    apply_database_link    => NULL,
    apply_tag              => NULL,
    apply_captured         => FALSE,
    precommit_handler      => NULL,
    negative_rule_set_name => NULL);
END;
/
```

   Running this procedure performs the following actions:

   - Creates an apply process named `strm03_apply`. An apply process with the same name must not exist.
   - Associates the apply process with the queue named `strmadmin.streams_queue`. This queue must exist.
   - Associates the apply process with the rule set `strmadmin.strm03_rule_set`. This rule set must exist. This rule set is the positive rule set for the apply process.
   - Specifies that the apply process does not use a message handler.
   - Specifies that the 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. See Chapter 10, "Oracle Streams Tags" for more information about tags.
- Specifies that the apply process does not apply captured LCRs. Therefore, the apply process can apply persistent LCRs or persistent user messages that are in the persistent queue portion of the apply process's queue.
- Specifies that the apply process does not use a precommit handler.
- Specifies that the apply process does not use a negative rule set.

After creating the apply process, run the `ADD_TABLE_RULES` or `ADD_SUBSET_RULES` procedure to add rules to the apply process rule set. These rules direct the apply process to apply LCRs for the specified tables.

> **See Also:** *Oracle Streams Concepts and Administration* for more information about rules

8

Instantiation and Oracle Streams Replication

This chapter contains conceptual information about instantiation and Oracle Streams replication. It also contains instructions for preparing database objects for instantiation, performing instantiations, setting instantiation system change numbers (SCNs), and monitoring instantiations.

This chapter contains these topics:

- Overview of Instantiation and Oracle Streams Replication
- Capture Rules and Preparation for Instantiation
- Oracle Data Pump and Oracle Streams Instantiation
- Recovery Manager (RMAN) and Oracle Streams Instantiation
- Setting Instantiation SCNs at a Destination Database
- Monitoring Instantiation

Overview of Instantiation and Oracle Streams Replication

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

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

1. Prepare the database object for instantiation at the source database.
2. If a copy of the database object does not exist at the destination database, then create a database object physically at the destination database based on a database object at the source database. You can use export/import, transportable tablespaces, or RMAN to copy database objects for instantiation. If the database object already exists at the destination database, then this step is not necessary.
3. Set the instantiation system change number (SCN) for the database object at the destination database. An instantiation SCN instructs an apply process at the

destination database to apply only changes that committed at the source database after the specified SCN.

All of these instantiation steps can be performed automatically when you use one of the following Oracle-supplied procedures in the DBMS_STREAMS_ADM package that configure replication environments:

- MAINTAIN_GLOBAL
- MAINTAIN_SCHEMAS
- MAINTAIN_SIMPLE_TTS
- MAINTAIN_TABLES
- MAINTAIN_TTS

In some cases, Step 1 and Step 3 are completed automatically. For example, when you add rules for a database object to the positive rule set of a capture process by running a procedure in the DBMS_STREAMS_ADM package, the database object is prepared for instantiation automatically.

Also, when you use export/import, transportable tablespaces, or the RMAN TRANSPORT TABLESPACE command to copy database objects from a source database to a destination database, instantiation SCNs can be set for these database objects automatically.

Note: The RMAN DUPLICATE command can instantiate an entire database, but this command does not set instantiation SCNs for database objects.

If the database object being instantiated is a table, then the tables at the source and destination database do not need to be an exact match. However, 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. Whenever you plan to replicate changes to a database object, you must always prepare the database object for instantiation at the source database and set the instantiation SCN for the database object at the destination database. By preparing an object for instantiation, you are setting the lowest SCN for which changes to the object can be applied at destination databases. This SCN is called the ignore SCN. You should prepare a database object for instantiation after a capture process or synchronous capture has been configured to capture changes to the database object.

When you instantiate tables using export/import, transportable tablespaces, or RMAN, any 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. However, the RMAN DUPLICATE command retains database supplemental logging specifications at the instantiated database.

Note:

- During an export for an Oracle Streams instantiation, ensure that no data definition language (DDL) changes are made to objects being exported.
- When you export a database or schema that contains rules with non-`NULL` action contexts, 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.

See Also:

- *Oracle Streams Concepts and Administration* for more information about the oldest SCN for an apply process
- "Configuring Replication Using the DBMS_STREAMS_ADM Package" on page 2-3
- "Specifying Supplemental Logging" on page 1-28 for information about adding and dropping supplemental log groups

Capture Rules and Preparation for Instantiation

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

- The `PREPARE_TABLE_INSTANTIATION` procedure prepares a single table for instantiation when changes to the table will be captured by a capture process.
- The `PREPARE_SYNC_INSTANTIATION` function prepares a single table or multiple tables for instantiation when changes to the table or tables will be captured by a synchronous capture.
- The `PREPARE_SCHEMA_INSTANTIATION` procedure prepares for instantiation all of the database objects in a schema and all database objects added to the schema in the future. This procedure should only be used when changes will be captured by a capture process.
- The `PREPARE_GLOBAL_INSTANTIATION` procedure prepares for instantiation all of the database objects in a database and all database objects added to the database in the future. This procedure should only be used when changes will be captured by a capture process.

These procedures record the lowest system change number (SCN) of each object for instantiation. SCNs after the lowest SCN for an object can be used for instantiating the object.

If you use a capture process to capture changes, then these procedures also populate the Oracle Streams data dictionary for the relevant capture processes, propagations, and apply processes that capture, propagate, or apply changes made to the table, schema, or database being prepared for instantiation. In addition, if you use a capture process to capture changes, then these procedures optionally can enable supplemental logging for key columns or all columns in the tables that are being prepared for instantiation.

Note: Replication with synchronous capture does not use the Oracle Streams data dictionary and does not require supplemental logging.

See Also:

- "Preparing Database Objects for Instantiation at a Source Database" on page 8-8
- "Procedures That Automatically Specify Supplemental Logging" on page 1-33
- *Oracle Streams Concepts and Administration* for more information about the Oracle Streams data dictionary

DBMS_STREAMS_ADM Package Procedures Automatically Prepare Objects

When you add rules to the positive rule set for a capture process or synchronous capture by running a procedure in the DBMS_STREAMS_ADM package, a procedure or function in the DBMS_CAPTURE_ADM package is run automatically on the database objects where changes will be captured. Table 8–1 lists which procedure or function is run in the DBMS_CAPTURE_ADM package when you run a procedure in the DBMS_STREAMS_ADM package.

Table 8–1 DBMS_CAPTURE_ADM Package Procedures That Are Run Automatically

When you run this procedure in the DBMS_STREAMS_ADM package	This procedure or function in the DBMS_CAPTURE_ADM package is run automatically
ADD_TABLE_RULES ADD_SUBSET_RULES	PREPARE_TABLE_INSTANTIATION when rules are added to a capture process rule set PREPARE_SYNC_INSTANTIATION when rules are added to a synchronous capture rule set
ADD_SCHEMA_RULES	PREPARE_SCHEMA_INSTANTIATION
ADD_GLOBAL_RULES	PREPARE_GLOBAL_INSTANTIATION

Multiple calls to prepare for instantiation are allowed. If you are using downstream capture, and the downstream capture process uses a database link from the downstream database to the source database, then the database objects are prepared for instantiation automatically when you run one of these procedures in the DBMS_STREAMS_ADM package. However, if the downstream capture process does not use a database link from the downstream database to the source database, then you must prepare the database objects for instantiation manually.

When capture process rules are created by the DBMS_RULE_ADM package instead of the DBMS_STREAMS_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 apply changes that result from the capture process rules with an apply process.

In addition, some procedures automatically run these procedures. For example, the DBMS_STREAMS_ADM.MAINTAIN_TABLES procedure automatically runs the ADD_TABLE_RULES procedure.

Note: A synchronous capture only captures changes based on rules created by the `ADD_TABLE_RULES` or `ADD_SUBSET_RULES` procedures.

See Also: "Configuring Replication Using the DBMS_STREAMS_ADM Package" on page 2-3

When Preparing for Instantiation Is Required

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

- One or more rules are added to the positive rule set for a capture process that instruct the capture process to capture changes made to the object.
- One or more conditions of rules in the positive 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 positive rule set for a propagation that instruct the propagation to propagate changes made to the object.
- One or more conditions of rules in the positive rule set for a propagation are modified to instruct the propagation to propagate changes made to the object.
- One or more rules are added to the positive rule set for an apply process that instruct the apply process to apply changes that were made to the object at the source database.
- One or more conditions of rules in the positive rule set for an apply process are modified to instruct the apply process to apply changes that were made to the object at the source database.

Whenever you remove, or modify the condition of, a capture process, propagation, or apply process rule for a database object that is in a negative rule set, you must run the appropriate procedure to prepare the database object for instantiation at the source database if any of the following conditions are met:

- One or more rules are removed from the negative rule set for a capture process to instruct the capture process to capture changes made to the object.
- One or more conditions of rules in the negative 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 removed from the negative rule set for a propagation to instruct the propagation to propagate changes made to the object.
- One or more conditions of rules in the negative rule set for a propagation are modified to instruct the propagation to propagate changes made to the object.
- One or more rules are removed from the negative rule set for an apply process to instruct the apply process to apply changes that were made to the object at the source database.
- One or more conditions of rules in the negative rule set for an apply process are modified to instruct the apply process to apply changes that were made to the object at the source database.

When any of these conditions are met for changes to a positive or negative rule set, you must prepare the relevant database objects for instantiation at the source database

to populate any relevant Oracle 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 Oracle 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 Oracle Streams data dictionary is populated with the information about the object when a capture process captures these redo entries, and any remote Oracle Streams data dictionaries are populated when the information is propagated to them.

Synchronous captures do not use Oracle Streams data dictionaries. However, when you are capturing changes to a database object with synchronous capture, you must prepare the database object for instantiation when you add rules for the database object to the synchronous capture rule set. Preparing the database object for instantiation is required when rules are added because it records the lowest SCN for instantiation for the database object. Preparing the database object for instantiation is not required when synchronous capture rules are modified, but modifications cannot change the database object name or schema in the rule condition.

See Also:

- "Preparing Database Objects for Instantiation at a Source Database" on page 8-8
- *Oracle Streams Concepts and Administration* for more information about the Oracle Streams data dictionary

Supplemental Logging Options During Preparation for Instantiation

If a replication environment uses a capture process to capture changes, then supplemental logging is required. Supplemental logging places additional column data into a redo log whenever an operation is performed. The procedures in the `DBMS_CAPTURE_ADM` package that prepare database objects for instantiation include `PREPARE_TABLE_INSTANTIATION`, `PREPARE_SCHEMA_INSTANTIATION`, and `PREPARE_GLOBAL_INSTANTIATION`. These procedures have a `supplemental_logging` parameter which controls the supplemental logging specifications for the database objects being prepared for instantiation.

Table 8–2 describes the values for the `supplemental_logging` parameter for each procedure.

Table 8–2 Supplemental Logging Options During Preparation for Instantiation

Procedure	supplemental_logging Parameter Setting	Description
PREPARE_TABLE_INSTANTIATION	keys	The procedure enables supplemental logging for primary key, unique key, bitmap index, and foreign key columns in the table being prepared for instantiation. The procedure places the logged columns for the table in three separate log groups: the primary key columns in an unconditional log group, the unique key columns and bitmap index columns in a conditional log group, and the foreign key columns in a conditional log group.
PREPARE_TABLE_INSTANTIATION	all	The procedure enables supplemental logging for all columns in the table being prepared for instantiation. The procedure places all of the columns for the table in an unconditional log group.
PREPARE_SCHEMA_INSTANTIATION	keys	The procedure enables supplemental logging for primary key, unique key, bitmap index, and foreign key columns in the tables in the schema being prepared for instantiation and for any table added to this schema in the future. Primary key columns are logged unconditionally. Unique key, bitmap index, and foreign key columns are logged conditionally.
PREPARE_SCHEMA_INSTANTIATION	all	The procedure enables supplemental logging for all columns in the tables in the schema being prepared for instantiation and for any table added to this schema in the future. The columns are logged unconditionally.
PREPARE_GLOBAL_INSTANTIATION	keys	The procedure enables database supplemental logging for primary key, unique key, bitmap index, and foreign key columns in the tables in the database being prepared for instantiation and for any table added to the database in the future. Primary key columns are logged unconditionally. Unique key, bitmap index, and foreign key columns are logged conditionally.
PREPARE_GLOBAL_INSTANTIATION	all	The procedure enables supplemental logging for all columns in all of the tables in the database being prepared for instantiation and for any table added to the database in the future. The columns are logged unconditionally.
Any Prepare Procedure	none	The procedure does not enable supplemental logging for any columns in the tables being prepared for instantiation.

If the `supplemental_logging` parameter is not specified when one of prepare procedures is run, then `keys` is the default. Some procedures in the `DBMS_STREAMS_ADM` package prepare tables for instantiation when they add rules to a positive capture process rule set. In this case, the default supplemental logging option, `keys`, is specified for the tables being prepared for instantiation.

Note:

- When `all` is specified for the `supplemental_logging` parameter, supplemental logging is not enabled for columns of the following types: LOB, `LONG`, `LONG RAW`, user-defined type, and Oracle-supplied type.
- Specifying `keys` for the `supplemental_logging` parameter does not enable supplemental logging of bitmap join index columns.
- Oracle Database 10*g* Release 2 introduced the `supplemental_logging` parameter for the prepare procedures. By default, running these procedures enables supplemental logging. Before this release, these procedures did not enable supplemental logging. If you remove an Oracle Streams environment, or if you remove certain database objects from an Oracle Streams environment, then you can also remove the supplemental logging enabled by these procedures to avoid unnecessary logging.

See Also:

- "Preparing Database Objects for Instantiation at a Source Database" on page 8-8
- "Specifying Supplemental Logging" on page 1-28
- "DBMS_STREAMS_ADM Package Procedures Automatically Prepare Objects" on page 8-4
- "Aborting Preparation for Instantiation at a Source Database" on page 8-13 for information about removing supplemental logging enabled by the prepare procedures
- *Oracle Database SQL Language Reference* for information about data types

Preparing Database Objects for Instantiation at a Source Database

If you use the `DBMS_STREAMS_ADM` package to create rules for a capture process or a synchronous capture, then any objects referenced in the system-created rules are prepared for instantiation automatically. If you use the `DBMS_RULE_ADM` package to create rules for a capture process, then you must prepare the database objects referenced in these rules for instantiation manually. In this case, you should prepare a database object for instantiation after a capture process has been configured to capture changes to the database object. Synchronous captures ignore rules created by the `DBMS_RULE_ADM` package.

See "Capture Rules and Preparation for Instantiation" on page 8-3 for information about the PL/SQL subprograms that prepare database objects for instantiation. 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 waits until the long running transaction is complete before it records the ignore SCN for the objects. The ignore SCN is the SCN below which changes to an object cannot be applied at destination databases. Query the `V$STREAMS_TRANSACTION` dynamic performance view to monitor long running transactions being processed by a capture process or apply process.

The following sections contain examples that prepare database objects for instantiation:

- Preparing Tables for Instantiation
- Preparing the Database Objects in a Schema for Instantiation
- Preparing All of the Database Objects in a Database for Instantiation

> **See Also:** *Oracle Streams Concepts and Administration* for more information about the instantiation SCN and ignore SCN for an apply process

Preparing Tables for Instantiation

This section contains these topics:

- Preparing a Table for Instantiation Using DBMS_STREAMS_ADM When a Capture Process Is Used
- Preparing a Table for Instantiation Using DBMS_CAPTURE_ADM When a Capture Process Is Used
- Preparing Tables for Instantiation Using DBMS_STREAMS_ADM When a Synchronous Capture Is Used
- Preparing Tables for Instantiation Using DBMS_CAPTURE_ADM When a Synchronous Capture Is Used

Preparing a Table for Instantiation Using DBMS_STREAMS_ADM When a Capture Process Is Used The example in this section prepares a table for instantiation using the `DBMS_STREAMS_ADM` package when a capture process captures changes to the table. To prepare the `hr.regions` table for instantiation and enable supplemental logging for any primary key, unique key, bitmap index, and foreign key columns in the table, add rules for the `hr.regions` table to the positive rule set for a capture process using a procedure in the `DBMS_STREAMS_ADM` package. If the capture process is a local capture process or a downstream capture process with a database link to the source database, then the procedure that you run prepares this table for instantiation automatically.

The following procedure adds rules to the positive rule set of a capture process named `strm01_capture` and prepares the `hr.regions` table for instantiation:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name     => 'hr.regions',
    streams_type   => 'capture',
    streams_name   => 'strm01_capture',
    queue_name     => 'strmadmin.strm01_queue',
    include_dml    => TRUE,
    include_ddl    => TRUE,
    inclusion_rule => TRUE);
END;
/
```

> **See Also:** "Specifying Supplemental Logging" on page 1-28

Preparing a Table for Instantiation Using DBMS_CAPTURE_ADM When a Capture Process Is Used The example in this section prepares a table for instantiation using the `DBMS_CAPTURE_ADM` package when a capture process captures changes to the table. To

prepare the hr.regions table for instantiation and enable supplemental logging for any primary key, unique key, bitmap index, and foreign key columns in the table, run the following procedure:

```
BEGIN
  DBMS_CAPTURE_ADM.PREPARE_TABLE_INSTANTIATION(
    table_name           => 'hr.regions',
    supplemental_logging => 'keys');
END;
/
```

The default value for the supplemental_logging parameter is keys. Therefore, if this parameter is not specified, then supplemental logging is enabled for any primary key, unique key, bitmap index, and foreign key columns in the table that is being prepared for instantiation.

See Also: "Specifying Supplemental Logging" on page 1-28

Preparing Tables for Instantiation Using DBMS_STREAMS_ADM When a Synchronous Capture Is Used The example in this section prepares all of the tables in the hr schema for instantiation using the DBMS_STREAMS_ADM package when a synchronous capture captures changes to the tables.Add rules for the hr.jobs_transport and hr.regions_transport tables to the positive rule set for a synchronous capture using a procedure in the DBMS_STREAMS_ADM package. The procedure that you run prepares the tables for instantiation automatically.

The following procedure adds a rule to the positive rule set of a synchronous capture named sync_capture and prepares the hr.regions table for instantiation:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_TABLE_RULES(
    table_name   => 'hr.regions',
    streams_type => 'sync_capture',
    streams_name => 'sync_capture',
    queue_name   => 'strmadmin.streams_queue');
END;
/
```

Preparing Tables for Instantiation Using DBMS_CAPTURE_ADM When a Synchronous Capture Is Used The example in this section prepares all of the tables in the hr schema for instantiation using the DBMS_CAPTURE_ADM package when a synchronous capture captures changes to the tables. To prepare the tables in the hr schema for instantiation, run the following function:

```
SET SERVEROUTPUT ON
DECLARE
  tables       DBMS_UTILITY.UNCL_ARRAY;
  prepare_scn  NUMBER;
  BEGIN
    tables(1) := 'hr.departments';
    tables(2) := 'hr.employees';
    tables(3) := 'hr.countries';
    tables(4) := 'hr.regions';
    tables(5) := 'hr.locations';
    tables(6) := 'hr.jobs';
    tables(7) := 'hr.job_history';
    prepare_scn := DBMS_CAPTURE_ADM.PREPARE_SYNC_INSTANTIATION(
                     table_names => tables);
  DBMS_OUTPUT.PUT_LINE('Prepare SCN = ' || prepare_scn);
```

```
END;
/
```

Preparing the Database Objects in a Schema for Instantiation

This section contains these topics:

- Preparing the Database Objects in a Schema for Instantiation Using DBMS_STREAMS_ADM
- Preparing the Database Objects in a Schema for Instantiation Using DBMS_CAPTURE_ADM

Preparing the Database Objects in a Schema for Instantiation Using DBMS_STREAMS_ADM The example in this section prepares the database objects in a schema for instantiation using the DBMS_STREAMS_ADM package when a capture process captures changes to these objects.

To prepare the database objects in the hr schema for instantiation and enable supplemental logging for the all columns in the tables in the hr schema, run the following procedure, add rules for the hr schema to the positive rule set for a capture process using a procedure in the DBMS_STREAMS_ADM package. If the capture process is a local capture process or a downstream capture process with a database link to the source database, then the procedure that you run prepares the objects in the hr schema for instantiation automatically.

The following procedure adds rules to the positive rule set of a capture process named strm01_capture and prepares the hr schema, and all of its database objects, for instantiation:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
    schema_name     =>  'hr',
    streams_type    =>  'capture',
    streams_name    =>  'strm01_capture',
    queue_name      =>  'strm01_queue',
    include_dml     =>  TRUE,
    include_ddl     =>  TRUE,
    inclusion_rule  =>  TRUE);
END;
/
```

If the specified capture process does not exist, then this procedure creates it.

In addition, supplemental logging is enabled for any primary key, unique key, bitmap index, and foreign key columns in the tables that are being prepared for instantiation.

> **See Also:** "Specifying Supplemental Logging" on page 1-28

Preparing the Database Objects in a Schema for Instantiation Using DBMS_CAPTURE_ADM The example in this section prepares the database objects in a schema for instantiation using the DBMS_CAPTURE_ADM package when a capture process captures changes to these objects. To prepare the database objects in the hr schema for instantiation and enable supplemental logging for the all columns in the tables in the hr schema, run the following procedure:

```
BEGIN
  DBMS_CAPTURE_ADM.PREPARE_SCHEMA_INSTANTIATION(
    schema_name          => 'hr',
    supplemental_logging => 'all');
END;
```

```
/
```

After running this procedure, supplemental logging is enabled for all of the columns in the tables in the hr schema and for all of the columns in the tables added to the hr schema in the future.

See Also: "Specifying Supplemental Logging" on page 1-28

Preparing All of the Database Objects in a Database for Instantiation

This section contains these topics:

- Preparing All of the Database Objects in a Database for Instantiation Using DBMS_STREAMS_ADM
- Preparing All of the Database Objects in a Database for Instantiation Using DBMS_CAPTURE_ADM

Preparing All of the Database Objects in a Database for Instantiation Using DBMS_STREAMS_ADM The example in this section prepares the database objects in a database for instantiation using the DBMS_STREAMS_ADM package when a capture process captures changes to these objects. To prepare all of the database objects in a database for instantiation, run the ADD_GLOBAL_RULES procedure in the DBMS_STREAMS_ADM package:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_GLOBAL_RULES(
    streams_type   => 'capture',
    streams_name   => 'capture_db',
    queue_name     => 'strmadmin.streams_queue',
    include_dml    => TRUE,
    include_ddl    => TRUE,
    inclusion_rule => TRUE);
END;
/
```

If the specified capture process does not exist, then this procedure creates it.

In addition, supplemental logging is enabled for any primary key, unique key, bitmap index, and foreign key columns in the tables that are being prepared for instantiation.

See Also: "Specifying Supplemental Logging" on page 1-28

Preparing All of the Database Objects in a Database for Instantiation Using DBMS_CAPTURE_ADM The example in this section prepares the database objects in a database for instantiation using the DBMS_CAPTURE_ADM package when a capture process captures changes to these objects. To prepare all of the database objects in a database for instantiation, run the following procedure:

```
BEGIN
  DBMS_CAPTURE_ADM.PREPARE_GLOBAL_INSTANTIATION(
    supplemental_logging => 'none');
END;
/
```

Because none is specified for the supplemental_logging parameter, this procedure does not enable supplemental logging for any columns. However, you can specify supplemental logging manually using an ALTER TABLE or ALTER DATABASE statement.

See Also: "Specifying Supplemental Logging" on page 1-28

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 and removes any supplemental logging enabled by the PREPARE_TABLE_INSTANTIATION procedure.
- ABORT_SYNC_INSTANTIATION reverses the effects of PREPARE_SYNC_INSTANTIATION
- ABORT_SCHEMA_INSTANTIATION reverses the effects of PREPARE_SCHEMA_INSTANTIATION and removes any supplemental logging enabled by the PREPARE_SCHEMA_INSTANTIATION and PREPARE_TABLE_INSTANTIATION procedures.
- ABORT_GLOBAL_INSTANTIATION reverses the effects of PREPARE_GLOBAL_INSTANTIATION and removes any supplemental logging enabled by the PREPARE_GLOBAL_INSTANTIATION, PREPARE_SCHEMA_INSTANTIATION, and PREPARE_TABLE_INSTANTIATION procedures.

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;
/
```

Oracle Data Pump and Oracle Streams Instantiation

The following sections contain information about performing Oracle Streams instantiations using Oracle Data Pump:

- Data Pump Export and Object Consistency
- Oracle Data Pump Import and Oracle Streams Instantiation
- Instantiating Objects Using Data Pump Export/Import

See Also:

- *Oracle Streams Concepts and Administration* for information about performing a full database export/import on a database using Oracle Streams
- *Oracle Database Utilities* for more information about Data Pump

Data Pump Export and Object Consistency

During export, Oracle Data Pump automatically uses Oracle Flashback to ensure that the exported data and the exported procedural actions for each database object are consistent to a single point in time. When you perform an instantiation in an Oracle Streams environment, some degree of consistency is required. Using the Data Pump Export utility is sufficient to ensure this consistency for Oracle Streams instantiations.

If you are using an export dump file for other purposes in addition to an Oracle Streams instantiation, and these other purposes have more stringent consistency requirements than those provided by Data Pump's default export, then you can use the Data Pump Export utility parameters `FLASHBACK_SCN` or `FLASHBACK_TIME` for Oracle Streams instantiations. For example, if an export includes objects with foreign key constraints, then more stringent consistency might be required.

Oracle Data Pump Import and Oracle Streams Instantiation

The following sections provide information about Oracle Data Pump import and Oracle Streams instantiation:

- Instantiation SCNs and Data Pump Imports
- Instantiation SCNs and Oracle Streams Tags Resulting from Data Pump Imports
- The STREAMS_CONFIGURATION Data Pump Import Utility Parameter

Instantiation SCNs and Data Pump Imports

During a Data Pump import, an instantiation SCN is set at the import database for each database object that was prepared for instantiation at the export database before the Data Pump export was performed. The instantiation SCN settings are based on metadata obtained during Data Pump export.

See Also: *Oracle Streams Concepts and Administration*

Instantiation SCNs and Oracle Streams Tags Resulting from Data Pump Imports

A Data Pump import session can set its Oracle Streams tag to the hexadecimal equivalent of `'00'` to avoid cycling the changes made by the import. Redo entries resulting from such an import have this tag value.

Whether the import session tag is set to the hexadecimal equivalent of `'00'` depends on the export that is being imported. Specifically, the import session tag is set to the hexadecimal equivalent of `'00'` in either of the following cases:

- The Data Pump export was in `FULL` or `SCHEMA` mode.
- The Data Pump export was in `TABLE` or `TABLESPACE` mode and at least one table included in the export was prepared for instantiation at the export database before the export was performed.

If neither one of these conditions is true for a Data Pump export that is being imported, then the import session tag is `NULL`.

Note:

- If you perform a network import using Data Pump, then an implicit export is performed in the same mode as the import. For example, if the network import is in schema mode, then the implicit export is in schema mode also.
- The import session tag is not set if the Data Pump import is performed in `TRANSPORTABLE TABLESPACE` mode. An import performed in this mode does not generate any redo data for the imported data. Therefore, setting the session tag is not required.

See Also: Chapter 10, "Oracle Streams Tags"

The STREAMS_CONFIGURATION Data Pump Import Utility Parameter

The STREAMS_CONFIGURATION Data Pump Import utility parameter specifies whether to import any general Oracle Streams metadata that is 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 if the information is present in the export dump file:

- ANYDATA queues and their queue tables
- Queue subscribers
- Advanced Queuing agents
- Rules, including their positive and negative rule sets and evaluation contexts. All rules are imported, including Oracle Streams rules and non-Oracle Streams rules. Oracle Streams rules are rules generated by the system when certain procedures in the DBMS_STREAMS_ADM package are run, while non-Oracle Streams rules are rules created using the DBMS_RULE_ADM package.

  If the STREAMS_CONFIGURATION parameter is set to n, then information about Oracle Streams rules is not imported into the following data dictionary views: ALL_STREAMS_RULES, ALL_STREAMS_GLOBAL_RULES, ALL_STREAMS_SCHEMA_RULES, ALL_STREAMS_TABLE_RULES, DBA_STREAMS_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, if the information is present in the export dump file; when the STREAMS_CONFIGURATION Import utility parameter is set to n, the import does not include the following information:

- Capture processes that capture local changes, 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 each positive and negative rule set used by the capture process
  - Capture user for the capture process
  - The time that the status of the capture process last changed. This information is recorded in the DBA_CAPTURE data dictionary view.
  - If the capture process disabled or aborted, then the error number and message of the error that was the cause. This information is recorded in the DBA_CAPTURE data dictionary view.

- Synchronous captures, including the following information for each synchronous capture:
  - Name of the synchronous capture
  - Queue owner and queue name of the queue used by the synchronous capture
  - Rule set owner and rule set name of each rule set used by the synchronous capture
  - Capture user for the synchronous capture
- 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 `ANYDATA` queue that is used by an Oracle Streams client, either started or stopped. Oracle Streams clients include capture processes, synchronous captures, propagations, apply process, and messaging clients. `ANYDATA` queues themselves are imported regardless of the `STREAMS_CONFIGURATION` Import utility 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 each positive and negative rule set used by the propagation
  - Oracle Scheduler jobs related to Oracle Streams propagations
- 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 each positive and negative rule set used by the apply process
  - Whether the apply process applies captured LCRs in a buffered queue or messages in a persistent queue
  - Apply user for the apply process
  - Message handler used by the apply process, if one exists
  - DDL handler used by the apply process, if one exists.
  - Precommit handler used by the apply process, if one exists
  - Tag value 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 (oldest SCN), apply time, and applied message create time
  - Apply errors
  - The time that the status of the apply process last changed. This information is recorded in the `DBA_APPLY` data dictionary view
  - If the apply process disabled or aborted, then the error number and message of the error that was the cause. This information is recorded in the `DBA_APPLY` data dictionary view.
- DML handlers (including both statement DML handlers and procedure 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
- Messaging clients, including the following information for each messaging client:
  - Name of the messaging client
  - Queue owner and queue name of the queue used by the messaging client
  - Rule set owner and rule set name of each positive and negative rule set used by the messaging client
  - Message notification settings
- Some data dictionary information about Oracle Streams rules. The rules themselves are imported regardless of the setting for the `STREAMS_CONFIGURATION` parameter.
- Data dictionary information about Oracle Streams administrators, messaging clients, message rules, extra attributes included in logical change records (LCRs) captured by a capture process or synchronous capture, and extra attributes used in message rules

> **Note:** Downstream capture processes are not included in an import regardless of the `STREAMS_CONFIGURATION` setting.

Instantiating Objects Using Data Pump Export/Import

The example in this section describes the steps required to instantiate objects in an Oracle Streams environment using Oracle Data Pump export/import. This example makes the following assumptions:

- You want to capture changes to all of the database objects in the `hr` schema at a source database and apply these changes at a separate destination database.
- The `hr` schema exists at the source database but does not exist at the destination database. For the purposes of this example, you can drop the `hr` user at the destination database using the following SQL statement:

```
DROP USER hr CASCADE;
```

The Data Pump import re-creates the user and the user's database objects at the destination database.

- You have configured an Oracle Streams administrator at the source database and the destination database named `strmadmin`. At each database, the Oracle Streams administrator is granted `DBA` role.

  Note: The example in this section uses the command line Data Pump utility. You can also use the `DBMS_DATAPUMP` package for Oracle Streams instantiations.

  See Also:

  - "Configuring an Oracle Streams Administrator on All Databases" on page 1-13
  - *Oracle Database Utilities* for more information about Data Pump
  - *Oracle Streams Extended Examples* for examples that use the `DBMS_DATAPUMP` package for Oracle Streams instantiations

Given these assumptions, complete the following steps to instantiate the `hr` schema using Data Pump export/import:

1. In SQL*Plus, connect to the source database as the Oracle Streams administrator.

   See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.

2. Create a directory object to hold the export dump file and export log file:

   ```
   CREATE DIRECTORY DPUMP_DIR AS '/usr/dpump_dir';
   ```

3. Prepare the database objects in the `hr` schema for instantiation. See "Preparing the Database Objects in a Schema for Instantiation" on page 8-11 for instructions.

4. While still connected to the source database as the Oracle Streams administrator, determine the current system change number (SCN) of the source database:

   ```
   SELECT DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER FROM DUAL;
   ```

   The SCN value returned by this query is specified for the `FLASHBACK_SCN` Data Pump export parameter in Step 5. Because the `hr` schema includes foreign key constraints between tables, the `FLASHBACK_SCN` export parameter, or a similar export parameter, must be specified during export. In this example, assume that the query returned `876606`.

   After you perform this query, ensure that no DDL changes are made to the objects being exported until after the export is complete.

5. On a command line, use Data Pump to export the `hr` schema at the source database.

   Perform the export by connecting as an administrative user who is granted `EXP_FULL_DATABASE` role. This user also must have `READ` and `WRITE` privilege on the directory object created in Step 2. This example connects as the Oracle Streams administrator `strmadmin`.

   The following is a sample Data Pump export command:

```
expdp strmadmin SCHEMAS=hr DIRECTORY=DPUMP_DIR DUMPFILE=hr_schema_dp.dmp
FLASHBACK_SCN=876606
```

See Also: *Oracle Database Utilities* for information about performing a Data Pump export

6. In SQL*Plus, connect to the destination database as the Oracle Streams administrator.
7. Create a directory object to hold the import dump file and import log file:

   ```
   CREATE DIRECTORY DPUMP_DIR AS '/usr/dpump_dir';
   ```

8. Transfer the Data Pump export dump file `hr_schema_dp.dmp` to the destination database. You can use the `DBMS_FILE_TRANSFER` package, binary FTP, or some other method to transfer the file to the destination database. After the file transfer, the export dump file should reside in the directory that corresponds to the directory object created in Step 7.
9. On a command line at the destination database, use Data Pump to import the export dump file `hr_schema_dp.dmp`. Ensure that no changes are made to the tables in the schema being imported at the destination database until the import is complete. Performing the import automatically sets the instantiation SCN for the `hr` schema and all of its database objects at the destination database.

   Perform the import by connecting as an administrative user who is granted `IMP_FULL_DATABASE` role. This user also must have `READ` and `WRITE` privilege on the directory object created in Step 7. This example connects as the Oracle Streams administrator `strmadmin`.

   The following is a sample import command:

   ```
   impdp strmadmin SCHEMAS=hr DIRECTORY=DPUMP_DIR DUMPFILE=hr_schema_dp.dmp
   ```

   Note: Any table supplemental log groups for the tables exported from the export database are retained when the tables are imported at the import database. You can drop these supplemental log groups if necessary.

   See Also: *Oracle Database Utilities* for information about performing a Data Pump import

Recovery Manager (RMAN) and Oracle Streams Instantiation

The RMAN `TRANSPORT TABLESPACE` command can instantiate a tablespace or set of tablespaces, and the RMAN `DUPLICATE` and `CONVERT DATABASE` commands can instantiate an entire database. Using RMAN for instantiation usually is faster than other instantiation methods.

The following sections contain information about using these RMAN commands for instantiation:

- Instantiating Objects in a Tablespace Using Transportable Tablespace or RMAN
- Instantiating an Entire Database Using RMAN

Instantiating Objects in a Tablespace Using Transportable Tablespace or RMAN

The RMAN TRANSPORT TABLESPACE command uses Data Pump and an RMAN-managed auxiliary instance to export the database objects in a tablespace or tablespace set while the tablespace or tablespace set remains online in the source database. RMAN automatically starts an auxiliary instance with a system-generated name. The RMAN TRANSPORT TABLESPACE command produces a Data Pump export dump file and data files for the tablespace or tablespaces.

You can use Data Pump to import the dump file at the destination database, or you can use the ATTACH_TABLESPACES procedure in the DBMS_STREAMS_TABLESPACE_ADM package to attach the tablespace or tablespaces to the destination database. Also, instantiation SCN values for the database objects in the tablespace or tablespaces are set automatically at the destination database when the tablespaces are imported or attached.

Note: The RMAN TRANSPORT TABLESPACE command does not support user-managed auxiliary instances.

The examples in this section describe the steps required to instantiate the database objects in a tablespace using transportable tablespace or RMAN. These instantiation options usually are faster than export/import. The following examples instantiate the database objects in a tablespace:

- "Instantiating Objects Using Transportable Tablespace" on page 8-21 uses the transportable tablespace feature to complete the instantiation. Data Pump exports the tablespace at the source database and imports the tablespace at the destination database. The tablespace is read-only during the export.
- "Instantiating Objects Using Transportable Tablespace From Backup With RMAN" on page 8-23 uses the RMAN TRANSPORT TABLESPACE command to generate a Data Pump export dump file and data files for a tablespace or set of tablespaces at the source database while the tablespace or tablespaces remain online. Either Data Pump import or the ATTACH_TABLESPACES procedure in the DBMS_STREAMS_TABLESPACE_ADM package can add the tablespace or tablespaces to the destination database.

These examples instantiate a tablespace set that includes a tablespace called jobs_tbs, and a tablespace called regions_tbs. To run the examples, connect to the source database in SQL*Plus as an administrative user and create the new tablespaces:

```
CREATE TABLESPACE jobs_tbs DATAFILE '/usr/oracle/dbs/jobs_tbs.dbf' SIZE 5 M;

CREATE TABLESPACE regions_tbs DATAFILE '/usr/oracle/dbs/regions_tbs.dbf' SIZE 5 M;
```

Place the new table hr.jobs_transport in the jobs_tbs tablespace:

```
CREATE TABLE hr.jobs_transport TABLESPACE jobs_tbs AS
  SELECT * FROM hr.jobs;
```

Place the new table hr.regions_transport in the regions_tbs tablespace:

```
CREATE TABLE hr.regions_transport TABLESPACE regions_tbs AS
  SELECT * FROM hr.regions;
```

Both of the examples make the following assumptions:

- You want to capture all of the changes to the `hr.jobs_transport` and `hr.regions_transport` tables at a source database and apply these changes at a separate destination database.
- The `hr.jobs_transport` table exists at a source database, and a single self-contained tablespace named `jobs_tbs` contains the table. The `jobs_tbs` tablespace is stored in a single data file named `jobs_tbs.dbf`.
- The `hr.regions_transport` table exists at a source database, and a single self-contained tablespace named `regions_tbs` contains the table. The `regions_tbs` tablespace is stored in a single data file named `regions_tbs.dbf`.
- The `jobs_tbs` and `regions_tbs` tablespaces do not contain data from any other schemas.
- The `hr.jobs_transport` table, the `hr.regions_transport` table, the `jobs_tbs` tablespace, and the `regions_tbs` tablespace do not exist at the destination database.
- You have configured an Oracle Streams administrator at both the source database and the destination database named `strmadmin`, and you have granted this Oracle Streams administrator `DBA` role at both databases.

  See Also:

  - "Checking for Consistency After Instantiation" on page 13-37
  - "Configuring an Oracle Streams Administrator on All Databases" on page 1-13
  - *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus

Instantiating Objects Using Transportable Tablespace

This example uses transportable tablespace to instantiate the database objects in a tablespace set. In addition to the assumptions listed in "Instantiating Objects in a Tablespace Using Transportable Tablespace or RMAN" on page 8-20, this example makes the following assumptions:

- The Oracle Streams administrator at the source database is granted the `EXP_FULL_DATABASE` role to perform the transportable tablespaces export. The `DBA` role is sufficient because it includes the `EXP_FULL_DATABASE` role. In this example, the Oracle Streams administrator performs the transportable tablespaces export.
- The Oracle Streams administrator at the destination database is granted the `IMP_FULL_DATABASE` role to perform the transportable tablespaces import. The `DBA` role is sufficient because it includes the `IMP_FULL_DATABASE` role. In this example, the Oracle Streams administrator performs the transportable tablespaces export.

  See Also: *Oracle Database Administrator's Guide* for more information about using transportable tablespaces and for information about limitations that might apply

Complete the following steps to instantiate the database objects in the `jobs_tbs` and `regions_tbs` tablespaces using transportable tablespace:

1. In SQL*Plus, connect to the source database as the Oracle Streams administrator.

See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.

2. Create a directory object to hold the export dump file and export log file:

```
CREATE DIRECTORY TRANS_DIR AS '/usr/trans_dir';
```

3. Prepare the `hr.jobs_transport` and `hr.regions_transport` tables for instantiation. See "Preparing Tables for Instantiation" on page 8-9 for instructions.

4. Make the tablespaces that contain the objects you are instantiating read-only. In this example, the `jobs_tbs` and `regions_tbs` tablespaces contain the database objects.

```
ALTER TABLESPACE jobs_tbs READ ONLY;

ALTER TABLESPACE regions_tbs READ ONLY;
```

5. On a command line, use the Data Pump Export utility to export the `jobs_tbs` and `regions_tbs` tablespaces at the source database using transportable tablespaces export parameters. The following is a sample export command that uses transportable tablespaces export parameters:

```
expdp strmadmin TRANSPORT_TABLESPACES=jobs_tbs, regions_tbs
DIRECTORY=TRANS_DIR DUMPFILE=tbs_ts.dmp
```

When you run the export command, ensure that you connect as an administrative user who was granted `EXP_FULL_DATABASE` role and has `READ` and `WRITE` privileges on the directory object.

> **See Also:** *Oracle Database Utilities* for information about performing an export

6. In SQL*Plus, connect to the destination database as the Oracle Streams administrator.

7. Create a directory object to hold the import dump file and import log file:

```
CREATE DIRECTORY TRANS_DIR AS '/usr/trans_dir';
```

8. Transfer the data files for the tablespaces and the export dump file `tbs_ts.dmp` to the destination database. You can use the `DBMS_FILE_TRANSFER` package, binary FTP, or some other method to transfer these files to the destination database. After the file transfer, the export dump file should reside in the directory that corresponds to the directory object created in Step 7.

9. On a command line at the destination database, use the Data Pump Import utility to import the export dump file `tbs_ts.dmp` using transportable tablespaces import parameters. Performing the import automatically sets the instantiation SCN for the `hr.jobs_transport` and `hr.regions_transport` tables at the destination database.

The following is an example import command:

```
impdp strmadmin DIRECTORY=TRANS_DIR DUMPFILE=tbs_ts.dmp
TRANSPORT_DATAFILES=/usr/orc/dbs/jobs_tbs.dbf,/usr/orc/dbs/regions_tbs.dbf
```

When you run the import command, ensure that you connect as an administrative user who was granted `IMP_FULL_DATABASE` role and has `READ` and `WRITE` privileges on the directory object.

See Also: *Oracle Database Utilities* for information about performing an import

10. If necessary, at both the source database and the destination database, connect as the Oracle Streams administrator and put the tablespaces into read/write mode:

```
ALTER TABLESPACE jobs_tbs READ WRITE;

ALTER TABLESPACE regions_tbs READ WRITE;
```

Note: Any table supplemental log groups for the tables exported from the export database are retained when tables are imported at the import database. You can drop these supplemental log groups if necessary.

See Also: "Checking for Consistency After Instantiation" on page 13-37

Instantiating Objects Using Transportable Tablespace From Backup With RMAN

The RMAN `TRANSPORT TABLESPACE` command uses Data Pump and an RMAN-managed auxiliary instance to export the database objects in a tablespace or tablespace set while the tablespace or tablespace set remains online in the source database. The RMAN `TRANSPORT TABLESPACE` command produces a Data Pump export dump file and data files, and you can use these files to perform a Data Pump import of the tablespace or tablespaces at the destination database. You can also use the `ATTACH_TABLESPACES` procedure in the `DBMS_STREAMS_TABLESPACE_ADM` package to attach the tablespace or tablespaces at the destination database.

In addition to the assumptions listed in "Instantiating Objects in a Tablespace Using Transportable Tablespace or RMAN" on page 8-20, this example makes the following assumptions:

- The source database is `tts1.example.com`.
- The destination database is `tts2.example.com`.

See Also: *Oracle Database Backup and Recovery User's Guide* for instructions on using the RMAN `TRANSPORT TABLESPACE` command

Complete the following steps to instantiate the database objects in the `jobs_tbs` and `regions_tbs` tablespaces using transportable tablespaces and RMAN:

1. Create a backup of the source database that includes the tablespaces being instantiated, if a backup does not exist. RMAN requires a valid backup for tablespace cloning. In this example, create a backup of the source database that includes the `jobs_tbs` and `regions_tbs` tablespaces if one does not exist.
2. In SQL*Plus, connect to the source database `tts1.example.com` as the Oracle Streams administrator.

   See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.
3. Optionally, create a directory object to hold the export dump file and export log file:

```
CREATE DIRECTORY SOURCE_DIR AS '/usr/db_files';
```

This step is optional because the RMAN TRANSPORT TABLESPACE command creates a directory object named STREAMS_DIROBJ_DPDIR on the auxiliary instance if the DATAPUMP DIRECTORY parameter is omitted when you run this command in Step 9.

4. Prepare the hr.jobs_transport and hr.regions_transport tables for instantiation. See "Preparing Tables for Instantiation" on page 8-9 for instructions.

5. Determine the until SCN for the RMAN TRANSPORT TABLESPACE command:

```
SET SERVEROUTPUT ON SIZE 1000000
DECLARE
  until_scn NUMBER;
BEGIN
  until_scn:= DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER;
      DBMS_OUTPUT.PUT_LINE('Until SCN: ' || until_scn);
END;
/
```

Make a note of the until SCN returned. You will use this number in Step 9. For this example, assume that the returned until SCN is 7661956.

Optionally, you can skip this step. In this case, do not specify the until clause in the RMAN TRANSPORT TABLESPACE command in Step 9. When no until clause is specified, RMAN uses the last archived redo log file to determine the until SCN automatically.

6. In SQL*Plus, connect to the source database tts1.net as an administrative user.

7. Archive the current online redo log:

```
ALTER SYSTEM ARCHIVE LOG CURRENT;
```

8. Start the RMAN client, and connect to the source database tts1.example.com as TARGET.

See *Oracle Database Backup and Recovery Reference* for more information about the RMAN CONNECT command

9. At the source database tts1.example.com, use the RMAN TRANSPORT TABLESPACE command to generate the dump file for the tablespace set:

```
RMAN> RUN
      {
        TRANSPORT TABLESPACE 'jobs_tbs', 'regions_tbs'
        UNTIL SCN 7661956
        AUXILIARY DESTINATION '/usr/aux_files'
        DATAPUMP DIRECTORY SOURCE_DIR
        DUMP FILE 'jobs_regions_tbs.dmp'
        EXPORT LOG 'jobs_regions_tbs.log'
        IMPORT SCRIPT 'jobs_regions_tbs_imp.sql'
        TABLESPACE DESTINATION '/orc/dbs';
      }
```

The TRANSPORT TABLESPACE command places the files in the following directories on the computer system that runs the source database:

- The directory that corresponds to the SOURCE_DIR directory object (/usr/db_files) contains the export dump file and export log file.

- The `/orc/dbs` directory contains the generated data files for the tablespaces and the import script. You use this script to complete the instantiation by attaching the tablespace at the destination database.

10. Modify the import script, if necessary. You might need to modify one or both of the following items in the script:

    - You might want to change the method used to make the exported tablespaces part of the destination database. The import script includes two ways to make the exported tablespaces part of the destination database: a Data Pump import command (`impdp`), and a script for attaching the tablespaces using the `ATTACH_TABLESPACES` procedure in the `DBMS_STREAMS_TABLESPACE_ADM` package.

      The default script uses the attach tablespaces method. The Data Pump import command is commented out. To use Data Pump import, remove the comment symbols (`/*` and `*/`) surrounding the impdp command, and either surround the attach tablespaces script with comments or remove the attach tablespaces script. The attach tablespaces script starts with `SET SERVEROUTPUT ON` and continues to the end of the file.

    - You might need to change the directory paths specified in the script. In Step 11, you will transfer the import script (`jobs_regions_tbs_imp.sql`), the Data Pump export dump file (`jobs_regions_tbs.dmp`), and the generated data file for each tablespace (`jobs_tbs.dbf` and `regions_tbs.dbf`) to one or more directories on the computer system running the destination database. Ensure that the directory paths specified in the script are the correct directory paths.

11. Transfer the import script (`jobs_regions_tbs_imp.sql`), the Data Pump export dump file (`jobs_regions_tbs.dmp`), and the generated data file for each tablespace (`jobs_tbs.dbf` and `regions_tbs.dbf`) to the destination database. You can use the `DBMS_FILE_TRANSFER` package, binary FTP, or some other method to transfer the file to the destination database. After the file transfer, these files should reside in the directories specified in the import script.

12. In SQL*Plus, connect to the destination database `tts2.example.com` as the Oracle Streams administrator.

13. Run the import script:

    ```
    SET ECHO ON
    SPOOL jobs_tbs_imp.out
    @jobs_tbs_imp.sql
    ```

    When the script completes, check the `jobs_tbs_imp.out` spool file to ensure that all actions finished successfully.

    See Also: "Checking for Consistency After Instantiation" on page 13-37

Instantiating an Entire Database Using RMAN

The Recovery Manager (RMAN) `DUPLICATE` command creates a copy of the target database in another location. The command uses an RMAN auxiliary instance to restore backups of the target database files and create a new database. In an Oracle Streams instantiation, the target database is the source database and the new database that is created is the destination database. The RMAN `DUPLICATE` command requires that the source and destination database run on the same platform.

The RMAN CONVERT DATABASE command generates the data files and an initialization parameter file for a new destination database on a different platform. It also generates a script that creates the new destination database. These files can instantiate an entire destination database that runs on a different platform than the source database but has the same endian format as the source database.

The RMAN DUPLICATE and CONVERT DATABASE commands do not set the instantiation SCN values for the database objects. The instantiation SCN values must be set manually during instantiation.

The examples in this section describe the steps required to instantiate an entire database using the RMAN DUPLICATE command or CONVERT DATABASE command. To use one of these RMAN commands for full database instantiation, complete the following general steps:

1. Copy the entire source database to the destination site using the RMAN command.
2. Remove the Oracle Streams configuration at the destination site using the REMOVE_STREAMS_CONFIGURATION procedure in the DBMS_STREAMS_ADM package.
3. Configure Oracle Streams destination site, including configuration of one or more apply processes to apply changes from the source database.

You can complete this process without stopping any running capture processes or propagations at the source database.

Follow the instructions in one of these sections:

- Instantiating an Entire Database on the Same Platform Using RMAN
- Instantiating an Entire Database on Different Platforms Using RMAN

Note:

- To configure an Oracle Streams replication environment that replicates all of the supported changes for an entire database, you can use the PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP procedures in the DBMS_STREAMS_ADM package. See "Configuring Two-Database Global Replication with Local Capture" on page 2-16 for instructions.
- Oracle recommends that you do not use RMAN for instantiation in an environment where distributed transactions are possible. Doing so can cause in-doubt transactions that must be corrected manually. Use export/import or transportable tablespaces for instantiation instead.

See Also: "Configuring an Oracle Streams Administrator on All Databases" on page 1-13 for information about configuring an Oracle Streams administrator

Instantiating an Entire Database on the Same Platform Using RMAN

The example in this section instantiates an entire database using the RMAN DUPLICATE command. The example makes the following assumptions:

- You want to capture all of the changes made to a source database named `dpx1.example.com`, propagate these changes to a separate destination database named `dpx2.example.com`, and apply these changes at the destination database.
- You have configured an Oracle Streams administrator at the source database named `strmadmin`. See "Configuring an Oracle Streams Administrator on All Databases" on page 1-13.
- The `dpx1.example.com` and `dpx2.example.com` databases run on the same platform.

> **See Also:** *Oracle Database Backup and Recovery User's Guide* for instructions about using the RMAN `DUPLICATE` command

Complete the following steps to instantiate an entire database using RMAN when the source and destination databases run on the same platform:

1. Create a backup of the source database if one does not exist. RMAN requires a valid backup for duplication. In this example, create a backup of `dpx1.example.com` if one does not exist.

   > **Note:** A backup of the source database is not necessary if you use the `FROM ACTIVE DATABASE` option when you run the RMAN `DUPLICATE` command. For large databases, the `FROM ACTIVE DATABASE` option requires significant network resources. This example does not use this option.

2. In SQL*Plus, connect to the source database `dpx1.example.com` as the Oracle Streams administrator.

   See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.

3. Create an `ANYDATA` queue to stage the changes from the source database if such a queue does not already exist. This queue will stage changes that will be propagated to the destination database after it has been configured.

   For example, the following procedure creates a queue named `streams_queue`:

   ```
   EXEC DBMS_STREAMS_ADM.SET_UP_QUEUE();
   ```

   Remain connected as the Oracle Streams administrator in SQL*Plus at the source database through Step 9.

4. Create a database link from `dpx1.example.com` to `dpx2.example.com`:

   ```
   CREATE DATABASE LINK dpx2.example.com CONNECT TO strmadmin
      IDENTIFIED BY password USING 'dpx2.example.com';
   ```

5. Create a propagation from the source queue at the source database to the destination queue at the destination database. The destination queue at the destination database does not exist yet, but creating this propagation ensures that logical change records (LCRs) enqueued into the source queue will remain staged there until propagation is possible. In addition to captured LCRs, the source queue will stage internal messages that will populate the Oracle Streams data dictionary at the destination database.

   The following procedure creates the `dpx1_to_dpx2` propagation:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_GLOBAL_PROPAGATION_RULES(
    streams_name            => 'dpx1_to_dpx2',
    source_queue_name       => 'strmadmin.streams_queue',
    destination_queue_name  => 'strmadmin.streams_queue@dpx2.example.com',
    include_dml             => TRUE,
    include_ddl             => TRUE,
    source_database         => 'dpx1.example.com',
    inclusion_rule          => TRUE,
    queue_to_queue          => TRUE);
END;
/
```

6. Stop the propagation you created in Step 5.

```
BEGIN
  DBMS_PROPAGATION_ADM.STOP_PROPAGATION(
    propagation_name  => 'dpx1_to_dpx2');
END;
/
```

7. Prepare the entire source database for instantiation, if it has not been prepared for instantiation previously. See "Preparing All of the Database Objects in a Database for Instantiation" on page 8-12 for instructions.

   If there is no capture process that captures all of the changes to the source database, then create this capture process using the `ADD_GLOBAL_RULES` procedure in the `DBMS_STREAMS_ADM` package. If the capture process is a local capture process or a downstream capture process with a database link to the source database, then running this procedure automatically prepares the entire source database for instantiation. If such a capture process already exists, then ensure that the source database has been prepared for instantiation by querying the `DBA_CAPTURE_PREPARED_DATABASE` data dictionary view.

8. If you created a capture process in Step 7, then start the capture process:

```
BEGIN
  DBMS_CAPTURE_ADM.START_CAPTURE(
    capture_name  => 'capture_db');
END;
/
```

9. Determine the until SCN for the RMAN `DUPLICATE` command:

```
SET SERVEROUTPUT ON SIZE 1000000
DECLARE
  until_scn NUMBER;
BEGIN
  until_scn:= DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER;
      DBMS_OUTPUT.PUT_LINE('Until SCN: ' || until_scn);
END;
/
```

   Make a note of the until SCN returned. You will use this number in Step 14. For this example, assume that the returned until SCN is `3050191`.

10. In SQL*Plus, connect to the source database `dpx1.example.com` as an administrative user.

    See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.

11. Archive the current online redo log:

```
ALTER SYSTEM ARCHIVE LOG CURRENT;
```

12. Prepare your environment for database duplication, which includes preparing the destination database as an auxiliary instance for duplication. See *Oracle Database Backup and Recovery User's Guide* for instructions.

13. Start the RMAN client, and connect to the source database `dpx1.example.com` as `TARGET` and to the destination database `dpx2.example.com` as `AUXILIARY`.

    See *Oracle Database Backup and Recovery Reference* for more information about the RMAN `CONNECT` command.

14. Use the RMAN `DUPLICATE` command with the `OPEN RESTRICTED` option to instantiate the source database at the destination database. The `OPEN RESTRICTED` option is required. This option enables a restricted session in the duplicate database by issuing the following SQL statement: `ALTER SYSTEM ENABLE RESTRICTED SESSION`. RMAN issues this statement immediately before the duplicate database is opened.

    You can use the `UNTIL SCN` clause to specify an SCN for the duplication. Use the until SCN determined in Step 9 for this clause. The until SCN specified for the RMAN `DUPLICATE` command must be higher than the SCN when the database was prepared for instantiation in Step 7. Also, archived redo logs must be available for the until SCN specified and for higher SCN values. Therefore, Step 11 archived the redo log containing the until SCN.

    Ensure that you use `TO` *`database_name`* in the `DUPLICATE` command to specify the name of the duplicate database. In this example, the duplicate database name is `dpx2`. Therefore, the `DUPLICATE` command for this example includes `TO dpx2`.

    The following is an example of an RMAN `DUPLICATE` command:

```
RMAN> RUN
      {
        SET UNTIL SCN 3050191;
        ALLOCATE AUXILIARY CHANNEL dpx2 DEVICE TYPE sbt;
        DUPLICATE TARGET DATABASE TO dpx2
        NOFILENAMECHECK
        OPEN RESTRICTED;
      }
```

    See Also: *Oracle Database Backup and Recovery Reference* for more information about the RMAN `DUPLICATE` command

15. At the destination database, connect as an administrative user in SQL*Plus and rename the database global name. After the RMAN `DUPLICATE` command, the destination database has the same global name as the source database.

```
ALTER DATABASE RENAME GLOBAL_NAME TO DPX2.EXAMPLE.COM;
```

16. At the destination database, connect as an administrative user in SQL*Plus and run the following procedure:

    Caution: Ensure that you are connected to the destination database, not the source database, when you run this procedure because it removes the local Oracle Streams configuration.

```
EXEC DBMS_STREAMS_ADM.REMOVE_STREAMS_CONFIGURATION();
```

> **Note:** Any supplemental log groups for the tables at the source database are retained at the destination database, and the `REMOVE_STREAMS_CONFIGURATION` procedure does not drop them. You can drop these supplemental log groups if necessary.

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

17. At the destination database, use the `ALTER SYSTEM` statement to disable the `RESTRICTED SESSION`:

```
ALTER SYSTEM DISABLE RESTRICTED SESSION;
```

18. At the destination database, connect as the Oracle Streams administrator. See "Configuring an Oracle Streams Administrator on All Databases" on page 1-13.

19. At the destination database, create the queue specified in Step 5.

    For example, the following procedure creates a queue named `streams_queue`:

```
EXEC DBMS_STREAMS_ADM.SET_UP_QUEUE();
```

20. At the destination database, configure the Oracle Streams environment.

> **Note:** Do not start any apply processes at the destination database until after you set the global instantiation SCN in Step 22.

21. At the destination database, create a database link from the destination database to the source database:

```
CREATE DATABASE LINK dpx1.example.com CONNECT TO strmadmin
   IDENTIFIED BY password USING 'dpx1.example.com';
```

    This database link is required because the next step runs the `SET_GLOBAL_INSTANTIATION_SCN` procedure with the recursive parameter set to `TRUE`.

22. At the destination database, set the global instantiation SCN for the source database. The RMAN `DUPLICATE` command duplicates the database up to one less than the SCN value specified in the `UNTIL SCN` clause. Therefore, you should subtract one from the until SCN value that you specified when you ran the `DUPLICATE` command in Step 14. In this example, the until SCN was set to `3050191`. Therefore, the instantiation SCN should be set to `3050191 - 1`, or `3050190`.

    For example, to set the global instantiation SCN to `3050190` for the `dpx1.example.com` source database, run the following procedure:

```
BEGIN
  DBMS_APPLY_ADM.SET_GLOBAL_INSTANTIATION_SCN(
    source_database_name   =>  'dpx1.example.com',
    instantiation_scn      =>  3050190,
    recursive              =>  TRUE);
END;
/
```

Notice that the `recursive` parameter is set to `TRUE` to set the instantiation SCN for all schemas and tables in the destination database.

23. At the destination database, you can start any apply processes that you configured.

24. At the source database, start the propagation you stopped in Step 6:

```
BEGIN
  DBMS_PROPAGATION_ADM.START_PROPAGATION(
    queue_name  => 'dpx1_to_dpx2');
END;
/
```

> **See Also:** "Checking for Consistency After Instantiation" on page 13-37

Instantiating an Entire Database on Different Platforms Using RMAN

The example in this section instantiates an entire database using the RMAN `CONVERT DATABASE` command. The example makes the following assumptions:

- You want to capture all of the changes made to a source database named `cvx1.example.com`, propagate these changes to a separate destination database named `cvx2.example.com`, and apply these changes at the destination database.
- You have configured an Oracle Streams administrator at the source database named `strmadmin`. See "Configuring an Oracle Streams Administrator on All Databases" on page 1-13.
- The `cvx1.example.com` and `cvx2.example.com` databases run on different platforms, and the platform combination is supported by the RMAN `CONVERT DATABASE` command. You can use the `DBMS_TDB` package to determine whether a platform combination is supported.

The RMAN `CONVERT DATABASE` command produces converted data files, an initialization parameter file (PFILE), and a SQL script. The converted data files and PFILE are for use with the destination database, and the SQL script creates the destination database on the destination platform.

> **See Also:** *Oracle Database Backup and Recovery User's Guide* for instructions about using the RMAN `CONVERT DATABASE` command

Complete the following steps to instantiate an entire database using RMAN when the source and destination databases run on different platforms:

1. Create a backup of the source database if one does not exist. RMAN requires a valid backup. In this example, create a backup of `cvx1.example.com` if one does not exist.

2. In SQL*Plus, connect to the source database `cvx1.example.com` as the Oracle Streams administrator.

   See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.

3. Create an `ANYDATA` queue to stage the changes from the source database if such a queue does not already exist. This queue will stage changes that will be propagated to the destination database after it has been configured.

   For example, the following procedure creates a queue named `streams_queue`:

```
EXEC DBMS_STREAMS_ADM.SET_UP_QUEUE();
```

Remain connected as the Oracle Streams administrator in SQL*Plus at the source database through Step 8.

4. Create a database link from `cvx1.example.com` to `cvx2.example.com`:

```
CREATE DATABASE LINK cvx2.example.com CONNECT TO strmadmin
   IDENTIFIED BY password USING 'cvx2.example.com';
```

5. Create a propagation from the source queue at the source database to the destination queue at the destination database. The destination queue at the destination database does not exist yet, but creating this propagation ensures that logical change records (LCRs) enqueued into the source queue will remain staged there until propagation is possible. In addition to captured LCRs, the source queue will stage internal messages that will populate the Oracle Streams data dictionary at the destination database.

   The following procedure creates the `cvx1_to_cvx2` propagation:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_GLOBAL_PROPAGATION_RULES(
    streams_name            => 'cvx1_to_cvx2',
    source_queue_name       => 'strmadmin.streams_queue',
    destination_queue_name  => 'strmadmin.streams_queue@cvx2.example.com',
    include_dml             => TRUE,
    include_ddl             => TRUE,
    source_database         => 'cvx1.example.com',
    inclusion_rule          => TRUE,
    queue_to_queue          => TRUE);
END;
/
```

6. Stop the propagation you created in Step 5.

```
BEGIN
  DBMS_PROPAGATION_ADM.STOP_PROPAGATION(
    propagation_name  => 'cvx1_to_cvx2');
END;
/
```

7. Prepare the entire source database for instantiation, if it has not been prepared for instantiation previously. See "Preparing All of the Database Objects in a Database for Instantiation" on page 8-12 for instructions.

   If there is no capture process that captures all of the changes to the source database, then create this capture process using the `ADD_GLOBAL_RULES` procedure in the `DBMS_STREAMS_ADM` package. If the capture process is a local capture process or a downstream capture process with a database link to the source database, then running this procedure automatically prepares the entire source database for instantiation. If such a capture process already exists, then ensure that the source database has been prepared for instantiation by querying the `DBA_CAPTURE_PREPARED_DATABASE` data dictionary view.

8. If you created a capture process in Step 7, then start the capture process:

```
BEGIN
  DBMS_CAPTURE_ADM.START_CAPTURE(
    capture_name  => 'capture_db');
END;
/
```

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

   See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.

10. Archive the current online redo log:

```
ALTER SYSTEM ARCHIVE LOG CURRENT;
```

11. Prepare your environment for database conversion, which includes opening the source database in read-only mode. Complete the following steps:

    a. If the source database is open, then shut it down and start it in read-only mode.

    b. Run the `CHECK_DB` and `CHECK_EXTERNAL` functions in the `DBMS_TDB` package. Check the results to ensure that the conversion is supported by the RMAN `CONVERT DATABASE` command.

       See Also: *Oracle Database Backup and Recovery User's Guide* for more information about these steps

12. Determine the current SCN of the source database:

```
SET SERVEROUTPUT ON SIZE 1000000
DECLARE
  current_scn NUMBER;
BEGIN
  current_scn:= DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER;
      DBMS_OUTPUT.PUT_LINE('Current SCN: ' || current_scn);
END;
/
```

    Make a note of the SCN value returned. You will use this number in Step 24. For this example, assume that the returned value is `46931285`.

13. Start the RMAN client, and connect to the source database `cvx1.example.com` as `TARGET`.

    See *Oracle Database Backup and Recovery Reference* for more information about the RMAN `CONNECT` command.

14. Run the `CONVERT DATABASE` command.

    Ensure that you use `NEW DATABASE` *`database_name`* in the `CONVERT DATABASE` command to specify the name of the destination database. In this example, the destination database name is `cvx2`. Therefore, the `CONVERT DATABASE` command for this example includes `NEW DATABASE cvx2`.

    The following is an example of an RMAN `CONVERT DATABASE` command for a destination database that is running on the `Linux IA (64-bit)` platform:

```
CONVERT DATABASE NEW DATABASE 'cvx2'
          TRANSPORT SCRIPT '/tmp/convertdb/transportscript.sql'
          TO PLATFORM 'Linux IA (64-bit)'
          DB_FILE_NAME_CONVERT '/home/oracle/dbs','/tmp/convertdb';
```

15. Transfer the data files, PFILE, and SQL script produced by the RMAN `CONVERT DATABASE` command to the computer system that will run the destination database.

16. On the computer system that will run the destination database, modify the SQL script so that the destination database always opens with restricted session enabled.

    The following is a sample script with the necessary modifications in bold font:

```
-- The following commands will create a new control file and use it
-- to open the database.
-- Data used by Recovery Manager will be lost.
-- The contents of online logs will be lost and all backups will
-- be invalidated. Use this only if online logs are damaged.

-- After mounting the created controlfile, the following SQL
-- statement will place the database in the appropriate
-- protection mode:
--  ALTER DATABASE SET STANDBY DATABASE TO MAXIMIZE PERFORMANCE

STARTUP NOMOUNT PFILE='init_00gd2lak_1_0.ora'
CREATE CONTROLFILE REUSE SET DATABASE "CVX2" RESETLOGS  NOARCHIVELOG
    MAXLOGFILES 32
    MAXLOGMEMBERS 2
    MAXDATAFILES 32
    MAXINSTANCES 1
    MAXLOGHISTORY 226
LOGFILE
  GROUP 1 '/tmp/convertdb/archlog1'  SIZE 25M,
  GROUP 2 '/tmp/convertdb/archlog2'  SIZE 25M
DATAFILE
  '/tmp/convertdb/systemdf',
  '/tmp/convertdb/sysauxdf',
  '/tmp/convertdb/datafile1',
  '/tmp/convertdb/datafile2',
  '/tmp/convertdb/datafile3'
CHARACTER SET WE8DEC
;

-- NOTE: This ALTER SYSTEM statement is added to enable restricted session.

ALTER SYSTEM ENABLE RESTRICTED SESSION;

-- Database can now be opened zeroing the online logs.
ALTER DATABASE OPEN RESETLOGS;

-- No tempfile entries found to add.
--

set echo off
prompt ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
prompt * Your database has been created successfully!
prompt * There are many things to think about for the new database. Here
prompt * is a checklist to help you stay on track:
prompt * 1. You may want to redefine the location of the directory objects.
prompt * 2. You may want to change the internal database identifier (DBID)
prompt *    or the global database name for this database. Use the
prompt *    NEWDBID Utility (nid).
prompt ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

SHUTDOWN IMMEDIATE
-- NOTE: This startup has the UPGRADE parameter.
-- It already has restricted session enabled, so no change is needed.
STARTUP UPGRADE PFILE='init_00gd2lak_1_0.ora'
```

```
@@ ?/rdbms/admin/utlirp.sql
SHUTDOWN IMMEDIATE
-- NOTE: The startup below is generated without the RESTRICT clause.
-- Add the RESTRICT clause.
STARTUP RESTRICT PFILE='init_00gd2lak_1_0.ora'
-- The following step will recompile all PL/SQL modules.
-- It may take serveral hours to complete.
@@ ?/rdbms/admin/utlrp.sql
set feedback 6;
```

Other changes to the script might be necessary. For example, the data file locations and PFILE location might need to be changed to point to the correct locations on the destination database computer system.

17. At the destination database, connect as an administrative user in SQL*Plus and run the following procedure:

    Caution: Ensure that you are connected to the destination database, not the source database, when you run this procedure because it removes the local Oracle Streams configuration.

    ```
    EXEC DBMS_STREAMS_ADM.REMOVE_STREAMS_CONFIGURATION();
    ```

    Note: Any supplemental log groups for the tables at the source database are retained at the destination database, and the `REMOVE_STREAMS_CONFIGURATION` procedure does not drop them. You can drop these supplemental log groups if necessary.

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

18. In SQL*Plus, connect to the destination database `cvx2.example.com` as the Oracle Streams administrator.

19. Drop the database link from the source database to the destination database that was cloned from the source database:

    ```
    DROP DATABASE LINK cvx2.example.com;
    ```

20. At the destination database, use the `ALTER SYSTEM` statement to disable the `RESTRICTED SESSION`:

    ```
    ALTER SYSTEM DISABLE RESTRICTED SESSION;
    ```

21. At the destination database, create the queue specified in Step 5.

    For example, the following procedure creates a queue named `streams_queue`:

    ```
    EXEC DBMS_STREAMS_ADM.SET_UP_QUEUE();
    ```

22. At the destination database, connect as the Oracle Streams administrator and configure the Oracle Streams environment. See "Configuring an Oracle Streams Administrator on All Databases" on page 1-13.

> **Note:** Do not start any apply processes at the destination database until after you set the global instantiation SCN in Step 24.

23. At the destination database, create a database link to the source database:

```
CREATE DATABASE LINK cvx1.example.com CONNECT TO strmadmin
   IDENTIFIED BY password USING 'cvx1.example.com';
```

This database link is required because the next step runs the `SET_GLOBAL_INSTANTIATION_SCN` procedure with the recursive parameter set to `TRUE`.

24. At the destination database, set the global instantiation SCN for the source database to the SCN value returned in Step 12.

For example, to set the global instantiation SCN to `46931285` for the `cvx1.example.com` source database, run the following procedure:

```
BEGIN
  DBMS_APPLY_ADM.SET_GLOBAL_INSTANTIATION_SCN(
    source_database_name   =>  'cvx1.example.com',
    instantiation_scn      =>  46931285,
    recursive              =>  TRUE);
END;
/
```

Notice that the `recursive` parameter is set to `TRUE` to set the instantiation SCN for all schemas and tables in the destination database.

25. At the destination database, you can start any apply processes that you configured.

26. At the source database, start the propagation you stopped in Step 6:

```
BEGIN
  DBMS_PROPAGATION_ADM.START_PROPAGATION(
    propagation_name  => 'cvx1_to_cvx2');
END;
/
```

> **See Also:** "Checking for Consistency After Instantiation" on page 13-37

Setting Instantiation SCNs at a Destination Database

An instantiation system change number (SCN) instructs an apply process at a destination database to apply changes that committed after a specific SCN at a source database. You can set instantiation SCNs in one of the following ways:

- Export the relevant database objects at the source database and import them at the destination database. In this case, the export/import creates the database objects at the destination database, populates them with the data from the source database, and sets the relevant instantiation SCNs. You can use Data Pump export/import for instantiations. See "Setting Instantiation SCNs Using Export/Import" on page 8-37 for information about the instantiation SCNs that are set for different types of export/import operations.
- Perform a metadata only export/import using Data Pump. If you use Data Pump export/import, then set the `CONTENT` parameter to `METADATA_ONLY` during export at the source database or import at the destination database, or both.

Instantiation SCNs are set for the database objects, but no data is imported. See "Setting Instantiation SCNs Using Export/Import" on page 8-37 for information about the instantiation SCNs that are set for different types of export/import operations.

- Use transportable tablespaces to copy the objects in one or more tablespaces from a source database to a destination database. An instantiation SCN is set for each schema in these tablespaces and for each database object in these tablespaces that was prepared for instantiation before the export. See "Instantiating Objects in a Tablespace Using Transportable Tablespace or RMAN" on page 8-20.
- 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 "Setting Instantiation SCNs Using the DBMS_APPLY_ADM Package" on page 8-38.

See Also: *Oracle Streams Concepts and Administration*

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. You can specify a more stringent degree of consistency by using an export parameter such as `FLASHBACK_SCN` or `FLASHBACK_TIME`.

The following sections describe how the instantiation SCNs are 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` procedure, `PREPARE_SYNC_INSTANTIATION` function, `PREPARE_SCHEMA_INSTANTIATION` procedure, or `PREPARE_GLOBAL_INSTANTIATION` procedure 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 for them 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 SCN 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 table 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.

Note:

- If a non-NULL instantiation SCN already exists for a database object at a destination database that performs an import, then the import updates the instantiation SCN for that database object.
- During an export for an Oracle Streams instantiation, ensure that no data definition language (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.

See Also:

- "Oracle Data Pump and Oracle Streams Instantiation" on page 8-13 and *Oracle Database Utilities* for information about using export/import
- Part I, "Configuring Oracle Streams Replication" for more information about performing export/import operations to set instantiation SCNs when configuring an Oracle Streams environment
- "Preparing Database Objects for Instantiation at a Source Database" on page 8-8

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 can set the recursive parameter to TRUE when you run this procedure to set the instantiation SCN for each table in the schema. Similarly, if you set the instantiation SCN for a database using SET_GLOBAL_INSTANTIATION_SCN, then you can set the recursive parameter to TRUE when you run this procedure to set the instantiation SCN for the schemas in the database and for each table owned by these schemas.

Note:

- If you set the `recursive` parameter to `TRUE` in the `SET_SCHEMA_INSTANTIATION_SCN` procedure or the `SET_GLOBAL_INSTANTIATION_SCN` procedure, then a database link from the destination database to the source database is required. This database link must have the same name as the global name of the source database and must be accessible to the user who executes the procedure.
- When setting an instantiation SCN for a database object, always specify the name of the schema and database object at the source database, even if a rule-based transformation or apply handler is configured to change the schema name or database object name.
- If a relevant instantiation SCN is not present, then an error is raised during apply.
- These procedures can set an instantiation SCN for changes captured by capture processes and synchronous captures.

Table 8–3 lists each procedure and the types of statements for which they set an instantiation SCN.

Table 8–3 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`

Setting the Instantiation SCN While Connected to the Source Database

The user who runs the examples in this section must have access to a database link from the source database to the destination database. In these examples, the database link is `hrdb2.example.com`. The following example sets the instantiation SCN for

the `hr.departments` table at the `hrdb2.example.com` database to the current SCN by running the following procedure at the source database `hrdb1.example.com`:

```
DECLARE
  iscn  NUMBER;         -- Variable to hold instantiation SCN value
BEGIN
  iscn := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER();
  DBMS_APPLY_ADM.SET_TABLE_INSTANTIATION_SCN@HRDB2.EXAMPLE.COM(
    source_object_name    => 'hr.departments',
    source_database_name  => 'hrdb1.example.com',
    instantiation_scn     => iscn);
END;
/
```

The following example sets the instantiation SCN for the `oe` schema and all of its objects at the `hrdb2.example.com` database to the current source database SCN by running the following procedure at the source database `hrdb1.example.com`:

```
DECLARE
  iscn  NUMBER;         -- Variable to hold instantiation SCN value
BEGIN
  iscn := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER();
  DBMS_APPLY_ADM.SET_SCHEMA_INSTANTIATION_SCN@HRDB2.EXAMPLE.COM(
    source_schema_name    => 'oe',
    source_database_name  => 'hrdb1.example.com',
    instantiation_scn     => iscn,
    recursive             => TRUE);
END;
/
```

Because the `recursive` parameter is set to `TRUE`, running this procedure sets the instantiation SCN for each database object in the `oe` schema.

Note: When you set the `recursive` parameter to `TRUE`, a database link from the destination database to the source database is required, even if you run the procedure while you are connected to the source database. This database link must have the same name as the global name of the source database and must be accessible to the current user.

Setting the Instantiation SCN While Connected to the Destination Database

The user who runs the examples in this section must have access to a database link from the destination database to the source database. In these examples, the database link is `hrdb1.example.com`. The following example sets the instantiation SCN for the `hr.departments` table at the `hrdb2.example.com` database to the current source database SCN at `hrdb1.example.com` by running the following procedure at the destination database `hrdb2.example.com`:

```
DECLARE
  iscn  NUMBER;         -- Variable to hold instantiation SCN value
BEGIN
  iscn := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER@HRDB1.EXAMPLE.COM;
  DBMS_APPLY_ADM.SET_TABLE_INSTANTIATION_SCN(
    source_object_name    => 'hr.departments',
    source_database_name  => 'hrdb1.example.com',
    instantiation_scn     => iscn);
END;
/
```

The following example sets the instantiation SCN for the `oe` schema and all of its objects at the `hrdb2.example.com` database to the current source database SCN at `hrdb1.example.com` by running the following procedure at the destination database `hrdb2.example.com`:

```
DECLARE
  iscn  NUMBER;         -- Variable to hold instantiation SCN value
BEGIN
  iscn := DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER@HRDB1.EXAMPLE.COM;
  DBMS_APPLY_ADM.SET_SCHEMA_INSTANTIATION_SCN(
    source_schema_name    => 'oe',
    source_database_name  => 'hrdb1.example.com',
    instantiation_scn     => iscn,
    recursive             => TRUE);
END;
/
```

Because the `recursive` parameter is set to `TRUE`, running this procedure sets the instantiation SCN for each database object in the `oe` schema.

Note: 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:

- Part I, "Configuring Oracle Streams Replication" for more information when to set instantiation SCNs when you are configuring an Oracle Streams environment
- *Oracle Streams Extended Examples* for detailed examples that uses the `SET_TABLE_INSTANTIATION_SCN` procedure
- The information about the `DBMS_APPLY_ADM` package in the *Oracle Database PL/SQL Packages and Types Reference* for more information about which instantiation SCN can be used for a DDL LCR

Monitoring Instantiation

The following sections contain queries that you can run to determine which database objects are prepared for instantiation at a source database and the instantiation SCN for database objects at a destination database:

- Determining Which Database Objects Are Prepared for Instantiation
- Determining the Tables for Which an Instantiation SCN Has Been Set

Determining Which Database Objects Are Prepared for Instantiation

See "Capture Rules and Preparation for Instantiation" on page 8-3 for information about preparing database objects for instantiation.

To determine which database objects have been prepared for instantiation, query the following data dictionary views:

- DBA_CAPTURE_PREPARED_TABLES
- DBA_SYNC_CAPTURE_PREPARED_TABS
- DBA_CAPTURE_PREPARED_SCHEMAS
- DBA_CAPTURE_PREPARED_DATABASE

For example, to list all of the tables that have been prepared for instantiation by the PREPARE_TABLE_INSTANTIATION procedure, 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 'Prepare SCN' FORMAT 99999999999
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:

```
                                                Time Ready for
Table Owner     Table Name            Prepare SCN 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 8-8

Determining the Tables for Which an Instantiation SCN Has Been Set

An instantiation SCN is set at a destination database. It controls which captured logical change records (LCRs) for a database object 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 LCRs can be captured by a capture process or a synchronous capture. See "Setting Instantiation SCNs at a Destination Database" on page 8-36.

To determine which database objects have a set instantiation SCN, query the following corresponding data dictionary views:

- DBA_APPLY_INSTANTIATED_OBJECTS
- DBA_APPLY_INSTANTIATED_SCHEMAS
- DBA_APPLY_INSTANTIATED_GLOBAL

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 A20
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 99999999999

SELECT SOURCE_DATABASE,
       SOURCE_OBJECT_OWNER,
       SOURCE_OBJECT_NAME,
       INSTANTIATION_SCN
  FROM DBA_APPLY_INSTANTIATED_OBJECTS
  WHERE APPLY_DATABASE_LINK IS NULL;
```

Your output looks similar to the following:

```
Source Database     Object Owner    Object Name     Instantiation SCN
-------------------- --------------- --------------- -----------------
DBS1.EXAMPLE.COM     HR              REGIONS                    196660
DBS1.EXAMPLE.COM     HR              COUNTRIES                  196660
DBS1.EXAMPLE.COM     HR              LOCATIONS                  196660
```

Note: You can also display instantiation SCNs for changes that are applied to remote non-Oracle databases. This query does not display these instantiation SCNs because it lists an instantiation SCN only if the `APPLY_DATABASE_LINK` column is `NULL`.

See Also: "Setting Instantiation SCNs at a Destination Database" on page 8-36

9

Oracle Streams Conflict Resolution

Some Oracle 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 an Oracle Streams Environment
- Conflict Types in an Oracle Streams Environment
- Conflicts and Transaction Ordering in an Oracle Streams Environment
- Conflict Detection in an Oracle Streams Environment
- Conflict Avoidance in an Oracle Streams Environment
- Conflict Resolution in an Oracle Streams Environment
- Managing Oracle Streams Conflict Detection and Resolution
- Monitoring Conflict Detection and Update Conflict Handlers

About DML Conflicts in an Oracle Streams Environment

A **conflict** is a mismatch between the old values in an LCR and the expected data in a table. Conflicts can occur in an Oracle Streams environment that permits concurrent data manipulation language (DML) operations on the same data at multiple databases. In an Oracle Streams environment, DML conflicts can occur only when an apply process is applying a message that contains a row change resulting from a DML operation. This type of message 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 at different databases update the same row at nearly the same time, a conflict can occur. When you configure an Oracle 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 an Oracle 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 messages. Ensure that your environment avoids these types of conflicts.

See Also: *Oracle Streams Concepts and Administration* for more information about row LCRs

Conflict Types in an Oracle Streams Environment

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

- Update Conflicts in an Oracle Streams Environment
- Uniqueness Conflicts in an Oracle Streams Environment
- Delete Conflicts in an Oracle Streams Environment
- Foreign Key Conflicts in an Oracle Streams Environment

Update Conflicts in an Oracle 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 an Oracle 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 an Oracle Streams Environment

A **delete conflict** occurs when two transactions originate at 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 an Oracle 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 at 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 an Oracle Streams Environment

Ordering conflicts can occur in an Oracle 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.example.com, mult2.example.com, and mult3.example.com. Suppose a change is made to a row in the hr.departments table at mult1.example.com that will be propagated to both mult2.example.com and mult3.example.com. The following series of actions might occur:

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

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

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 the error queue after the required data has been propagated to the remote database and applied.

Conflict Detection in an Oracle Streams Environment

An apply process detects update, uniqueness, delete, and foreign key 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 can 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 can also be detected when either the `EXECUTE_ERROR` or `EXECUTE_ALL_ERRORS` procedure in the `DBMS_APPLY_ADM` package is run.

Note:

- If a column is updated and the column's old value equals its new value, then Oracle never detects a conflict for this column update.
- Any old LOB values in update LCRs, delete LCRs, and LCRs dealing with piecewise updates to LOB columns are not used by conflict detection.

Control Over Conflict Detection for Nonkey Columns

By default, an apply process compares old values for all columns during conflict detection, but you can stop conflict detection for nonkey columns using the `COMPARE_OLD_VALUES` procedure in the `DBMS_APPLY_ADM` package. Conflict detection might not be needed for some nonkey columns.

See Also:

- "Stopping Conflict Detection for Nonkey Columns" on page 9-14
- "Displaying Information About Conflict Detection" on page 9-16

Rows Identification During Conflict Detection in an Oracle 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 a table uniquely. When a table does not have a primary key, you should 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: *Oracle Streams Concepts and Administration*

Conflict Avoidance in an Oracle 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 might be possible. Even so, typically you can use some simple strategies to avoid specific types of conflicts.

Avoid Uniqueness Conflicts in an Oracle 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 an Oracle 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:

Table 9–1 Customized Sequences for Oracle Streams Replication Environments

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 an Oracle 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 an Oracle 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 an Oracle Streams Environment

After an update conflict has been detected, a conflict handler can attempt to resolve it. Oracle 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 procedure DML handler or an error handler.

Whether you use 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 the error queue. You might 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 the error 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:

- *Oracle Streams Concepts and Administration* for more information about procedure DML handlers, error handlers, and the error queue
- *Oracle Database 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 Oracle Streams Conflict Detection and Resolution" on page 9-12 for instructions on adding, modifying, and removing an update conflict handler
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `SET_UPDATE_CONFLICT_HANDLER` procedure
- "Column Lists" on page 9-9
- "Resolution Columns" on page 9-10

Types of Prebuilt Update Conflict Handlers

Oracle provides the following types of prebuilt update conflict handlers for an Oracle 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.example.com` 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.example.com` destination database:

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

3. A capture process or synchronous capture captures the update at the `dbs1.example.com` source database and puts the resulting row LCR in a queue.
4. A propagation propagates the row LCR from the queue at `dbs1.example.com` to a queue at `dbs2.example.com`.
5. An apply process at `dbs2.example.com` attempts to apply the row LCR to the `hr.employees` table but encounters a conflict because the salary value at `dbs2.example.com` 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 `dbs2.example.com` destination database in the conflict example, then the new value in the row LCR overwrites the value at `dbs2.example.com`. 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 `dbs2.example.com` 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 `dbs2.example.com`.

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 dbs2.example.com 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 dbs2.example.com.

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. 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.

```
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 might 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 ensure that the trigger's firing property is fire once, which is the default. Otherwise, a new time might be marked when transactions are applied by an apply process, resulting in the loss of the actual time of the transaction.

See Also: *Oracle Streams Concepts and Administration*

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.example.com 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 multiple update conflict handlers 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 in the `hr.employees` table, 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 the error 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 the error 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 are multiple columns 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 custom

rule-based transformation can combine multiple columns from the source database into a single column in the column list at the destination database. For example, a custom rule-based transformation can 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 are multiple columns in a column list. For example, an apply handler or custom rule-based transformation can separate one address column from the source database into multiple columns that are in a column list at the destination database. A custom rule-based transformation can 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, `LONG`, `LONG RAW`, user-defined type, and Oracle-supplied type columns. Therefore, you should not include these types of columns in the `column_list` parameter when running the `SET_UPDATE_CONFLICT_HANDLER` procedure.

See Also:

- "Specifying Supplemental Logging" on page 1-28
- *Oracle Database SQL Language Reference* for information about data types

Resolution Columns

The **resolution column** is the column used to identify a prebuilt 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 the error 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 ensure that 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, if successive time stamps 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, 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`. You can also set the `operation_name` parameter to `DEFAULT` so that the handler is used by default for all operations.
- 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 procedure 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 procedure 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 the error 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:

- *Oracle Streams Concepts and Administration* for more information about managing error handlers
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the SET_DML_HANDLER procedure

Managing Oracle Streams Conflict Detection and Resolution

This section describes the following tasks:

- Setting an Update Conflict Handler
- Modifying an Existing Update Conflict Handler
- Removing an Existing Update Conflict Handler
- Stopping Conflict Detection for Nonkey Columns

See Also: "Displaying Information About Update Conflict Handlers" on page 9-17

Setting an Update Conflict Handler

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 an Oracle Streams environment captures changes to the hr.jobs table at dbs1.example.com and propagates these changes to the dbs2.example.com 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.example.com database should always overwrite the change at the dbs2.example.com database. In this environment, you can accomplish this goal by specifying an OVERWRITE handler at the dbs2.example.com database.

To specify an update conflict handler for the hr.jobs table in the hr schema at the dbs2.example.com database, run the following procedure at dbs2.example.com:

```
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;
/
```

All apply processes running on a database that apply changes to the specified table locally use the specified update conflict handler.

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 `dbs1.example.com` database.
- Prebuilt update conflict handlers do not support LOB, `LONG`, `LONG RAW`, user-defined type, and Oracle-supplied type columns. Therefore, you should not include these types of columns in the `column_list` parameter when running the procedure `SET_UPDATE_CONFLICT_HANDLER`.

See Also:

- "Specifying Supplemental Logging" on page 1-28
- *Oracle Streams Extended Examples* for an example Oracle Streams environment that illustrates using the `MAXIMUM` prebuilt method for time-based conflict resolution
- *Oracle Database SQL Language Reference* for information about data types

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 9-12, 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, to change the resolution column for an update conflict handler, you must remove and re-create the handler.

For example, suppose the environment changes, and you want changes from `dbs1.example.com` to be discarded in the event of a conflict, whereas previously changes from `dbs1.example.com` overwrote changes at `dbs2.example.com`. You can accomplish this goal by specifying a `DISCARD` handler at the `dbs2.example.com` database.

To modify the existing update conflict handler for the `hr.jobs` table in the `hr` schema at the `dbs2.example.com` 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 a 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 9-12 and then modified in "Modifying an Existing Update Conflict Handler" on page 9-13. 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;
/
```

Stopping Conflict Detection for Nonkey Columns

You can stop conflict detection for nonkey columns using the `COMPARE_OLD_VALUES` procedure in the `DBMS_APPLY_ADM` package.

For example, suppose you configure a `time` column for conflict resolution for the `hr.employees` table, as described in "MAXIMUM" on page 9-7. In this case, you can decide to stop conflict detection for the other nonkey columns in the table. After adding the `time` column and creating the trigger as described in that section, add the columns in the `hr.employees` table to the column list for an update conflict handler:

```
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;
/
```

This example does not include the primary key for the table in the column list because it assumes that the primary key is never updated. However, other key columns are included in the column list.

To stop conflict detection for all nonkey columns in the table for both `UPDATE` and `DELETE` operations at a destination database, run the following procedure:

```
DECLARE
  cols DBMS_UTILITY.LNAME_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';
  DBMS_APPLY_ADM.COMPARE_OLD_VALUES(
    object_name  => 'hr.employees',
    column_table => cols,
    operation    => '*',
    compare      => FALSE);
END;
/
```

The asterisk (`*`) specified for the `operation` parameter means that conflict detection is stopped for both `UPDATE` and `DELETE` operations. After you run this procedure, all apply processes running on the database that apply changes to the specified table locally do not detect conflicts on the specified columns. Therefore, in this example, the `time` column is the only column used for conflict detection.

> **Note:** The example in this section sets an update conflict handler before stopping conflict detection for nonkey columns. However, an update conflict handler is not required before you stop conflict detection for nonkey columns.

See Also:

- "Control Over Conflict Detection for Nonkey Columns" on page 9-4
- "Displaying Information About Conflict Detection" on page 9-16
- *Oracle Streams Extended Examples* for a detailed example that uses time-based conflict resolution
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `COMPARE_OLD_VALUES` procedure

Monitoring Conflict Detection and Update Conflict Handlers

The following sections contain queries that you can run to monitor an apply process in a Stream replication environment:

- Displaying Information About Conflict Detection
- Displaying Information About Update Conflict Handlers

See Also: *Oracle Streams Concepts and Administration*

Displaying Information About Conflict Detection

You can stop conflict detection for nonkey columns using the `COMPARE_OLD_VALUES` procedure in the `DBMS_APPLY_ADM` package. When you use this procedure, conflict detection is stopped for updates and deletes on the specified columns for all apply processes at a destination database. To display each column for which conflict detection has been stopped, run the following query:

```
COLUMN OBJECT_OWNER HEADING 'Table Owner' FORMAT A15
COLUMN OBJECT_NAME HEADING 'Table Name' FORMAT A20
COLUMN COLUMN_NAME HEADING 'Column Name' FORMAT A20
COLUMN COMPARE_OLD_ON_DELETE HEADING 'Compare|Old On|Delete' FORMAT A7
COLUMN COMPARE_OLD_ON_UPDATE HEADING 'Compare|Old On|Update' FORMAT A7

SELECT OBJECT_OWNER,
       OBJECT_NAME,
       COLUMN_NAME,
       COMPARE_OLD_ON_DELETE,
       COMPARE_OLD_ON_UPDATE
  FROM DBA_APPLY_TABLE_COLUMNS
  WHERE APPLY_DATABASE_LINK IS NULL;
```

Your output should look similar to the following:

```
                                                      Compare Compare
                                                      Old On  Old On
Table Owner     Table Name           Column Name      Delete  Update
--------------- -------------------- -------------------- ------- -------
HR              EMPLOYEES            COMMISSION_PCT       NO      NO
HR              EMPLOYEES            EMAIL                NO      NO
HR              EMPLOYEES            FIRST_NAME           NO      NO
HR              EMPLOYEES            HIRE_DATE            NO      NO
HR              EMPLOYEES            JOB_ID               NO      NO
HR              EMPLOYEES            LAST_NAME            NO      NO
HR              EMPLOYEES            PHONE_NUMBER         NO      NO
HR              EMPLOYEES            SALARY               NO      NO
```

Note: You can also stop conflict detection for changes that are applied to remote non-Oracle databases. This query does not display such specifications because it lists a specification only if the `APPLY_DATABASE_LINK` column is `NULL`.

See Also:

- "Control Over Conflict Detection for Nonkey Columns" on page 9-4
- "Stopping Conflict Detection for Nonkey Columns" on page 9-14

Displaying Information About Update Conflict Handlers

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                           Resolution
Owner Table Name   Method       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: "Managing Oracle Streams Conflict Detection and Resolution" on page 9-12

10

Oracle Streams Tags

This chapter explains the concepts related to Oracle Streams tags.

This chapter contains these topics:

- Introduction to Tags
- Tags and Rules Created by the DBMS_STREAMS_ADM Package
- Tags and Online Backup Statements
- Tags and an Apply Process
- Oracle Streams Tags in a Replication Environment
- Managing Oracle Streams Tags
- Monitoring Oracle Streams Tags

See Also: "Managing Oracle Streams Tags" on page 10-16

Introduction to Tags

Every redo entry in the redo log has a **tag** associated with it. The data type 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. The size limit for a tag value is 2000 bytes.

You can configure how tag values are interpreted. For example, you can use a tag 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 can 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).

Based on the rules in the rule sets for a capture process, the tag value in the redo entry for a change can determine whether the change is captured. Based on the rules in the

rule sets for a synchronous capture, the session tag value for a change can determine whether the change is captured. The tags become part of the LCRs captured by a capture process or synchronous capture.

Similarly, once a tag is part of an LCR, the value of the tag can determine whether a propagation propagates the LCR and whether an apply process applies the LCR. The behavior of a custom rule-based transformation or apply 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 in a custom rule-based transformation or an apply handler that uses a PL/SQL procedure. You cannot set a tag value for an existing LCR in a statement DML handler or change handler.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `SET_TAG` member procedure for LCRs
- *Oracle Streams Concepts and Administration* for more information about how rules are used in Oracle Streams

Tags and Rules Created by the DBMS_STREAMS_ADM Package

When you use a procedure in the `DBMS_STREAMS_ADM` package to create rules and set the `include_tagged_lcr` parameter to `FALSE`, 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 positive rule set with a single rule and assume the rule contains such a condition. In this case, Oracle Streams capture processes, synchronous captures, propagations, and apply processes behave in the following way:

- A capture process captures a change only if the tag in the redo log entry for the change is `NULL` and the rest of the rule conditions evaluate to `TRUE` for the change.
- A synchronous capture captures a change only if the tag for the session that makes the change is `NULL` and the rest of the rule conditions evaluate to `TRUE` for the change.
- A propagation propagates 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 LCR only if the tag in the LCR is `NULL` and the rest of the rule conditions evaluate to `TRUE` for the LCR.

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

- A capture process discards a change only if the tag in the redo log entry for the change is `NULL` and the rest of the rule conditions evaluate to `TRUE` for the change.
- A propagation or apply process discards LCR only if the tag in the LCR is `NULL` and the rest of the rule conditions evaluate to `TRUE` for the LCR.

In most cases, specify `TRUE` for the `include_tagged_lcr` parameter if rules are being added to a negative rule set so that changes are discarded regardless of their tag values.

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_PROPAGATION_RULES
- ADD_SUBSET_RULES
- ADD_TABLE_PROPAGATION_RULES
- ADD_TABLE_RULES

If you do not want the 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 database change does not depend on the value of the tag.

For example, consider a table rule that evaluates to TRUE for all DML changes to the hr.locations table that originated at the dbs1.example.com 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.example.com',
    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.EXAMPLE.COM' )
```

If a capture process uses a positive 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.example.com',
    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.EXAMPLE.COM' )
```

Notice that there is no condition relating to the tag. If a capture process uses a positive 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.

To modify the is_null_tag condition in an existing system-created rule, use an appropriate procedure in the DBMS_STREAMS_ADM package to create a rule that is the same as the rule you want to modify, except for the is_null_tag condition. Next, use the REMOVE_RULE procedure in the DBMS_STREAMS_ADM package to remove the old rule from the appropriate rule set. In addition, you can use the and_condition parameter for the procedures that create rules in the DBMS_STREAMS_ADM package to add conditions relating to tags to system-created rules.

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.

See Also:

- *Oracle Streams Concepts and Administration* for examples of rules generated by the procedures in the DBMS_STREAMS_ADM package
- *Oracle Database 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 an Apply Process" on page 10-18 for more information about the SET_TAG procedure

Tags and Online Backup Statements

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. An online backup statement uses the BEGIN BACKUP and END BACKUP clauses in an ALTER TABLESPACE or ALTER DATABASE statement.

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.

Note: If you use Recovery Manager (RMAN) to perform an online backup, then the online backup statements are not used, and there is no need to set Oracle Streams tags for backups.

See Also: *Oracle Database Backup and Recovery User's Guide* for information about making backups

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 can 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 the 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 an LCR. You can alter the tag value for an existing apply process using the `ALTER_APPLY` procedure in the `DBMS_APPLY_ADM` package.

Redo entries generated by an apply handler for an apply process have the tag value of the apply process, unless the handler sets the tag to a different value using the `SET_TAG` procedure. If a procedure 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:

- *Oracle Streams Concepts and Administration* for more information about the apply process
- "Tags and Rules Created by the DBMS_STREAMS_ADM Package" on page 10-2 for more information about the default tag condition in Oracle Streams rules
- "Managing Oracle Streams Tags" on page 10-16
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_STREAMS_ADM` package and the `DBMS_APPLY_ADM` package

Oracle Streams Tags in a Replication Environment

In an Oracle Streams environment that includes multiple databases 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, Oracle Streams is designed to avoid change cycling.

Using tags and appropriate rules for Oracle Streams capture processes, synchronous captures, propagations, and apply processes, you can avoid such change cycles. This section describes common Oracle Streams environments and how you can use tags and rules to avoid change cycling in these environments.

This section contains these topics:

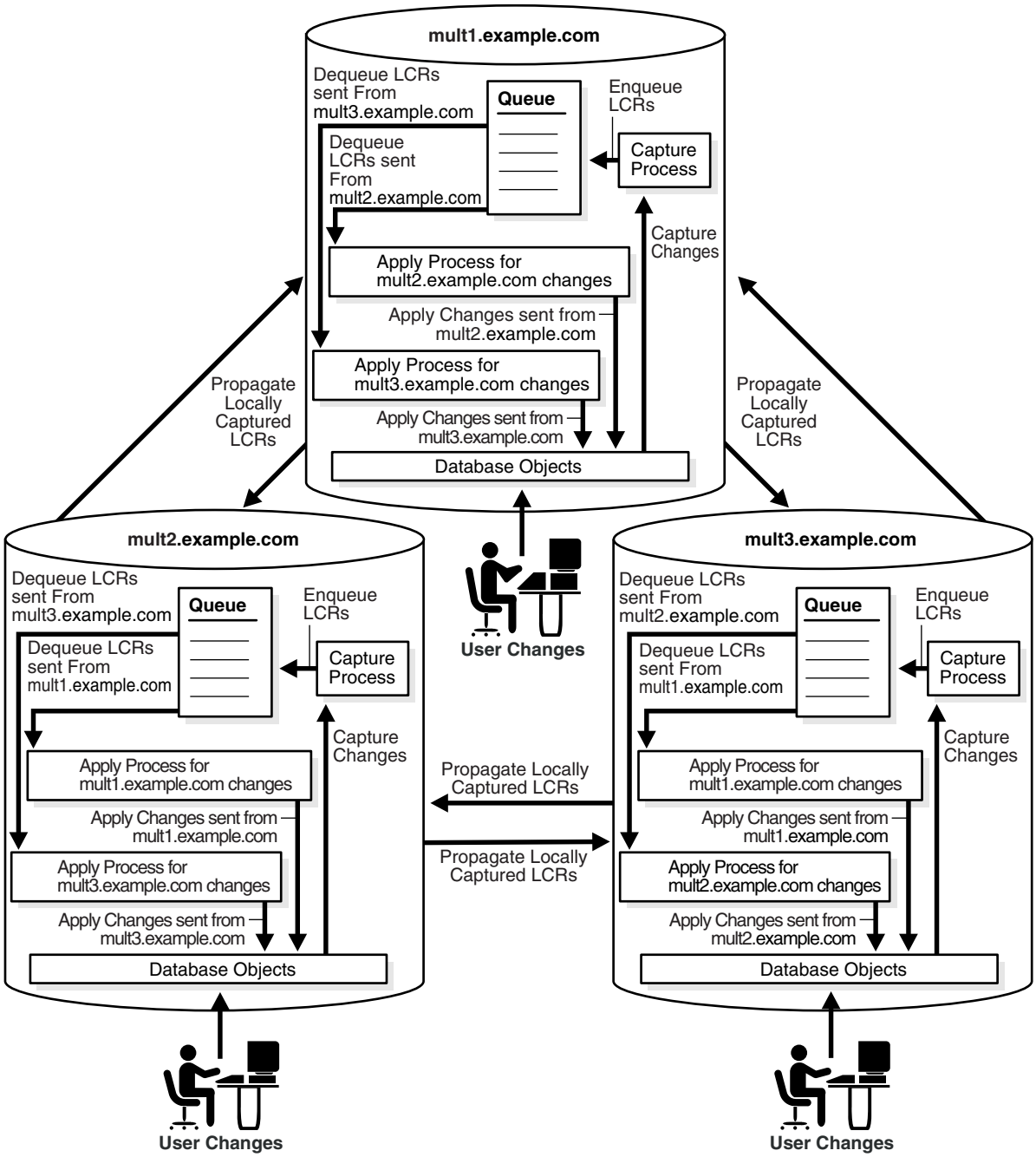
- N-Way Replication Environments
- Hub-and-Spoke Replication Environments
- Hub-and-Spoke Replication Environment with Several Extended Secondary Databases

N-Way Replication Environments

An **n-way replication** environment is one 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 `hrmult` schema between three Oracle databases: `mult1.example.com`, `mult2.example.com`, and `mult3.example.com`. DML and DDL changes made to tables in the `hrmult` 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 10–1 illustrates a sample n-way replication environment.

Figure 10–1 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 positive rule set used by the capture process has the following condition:

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

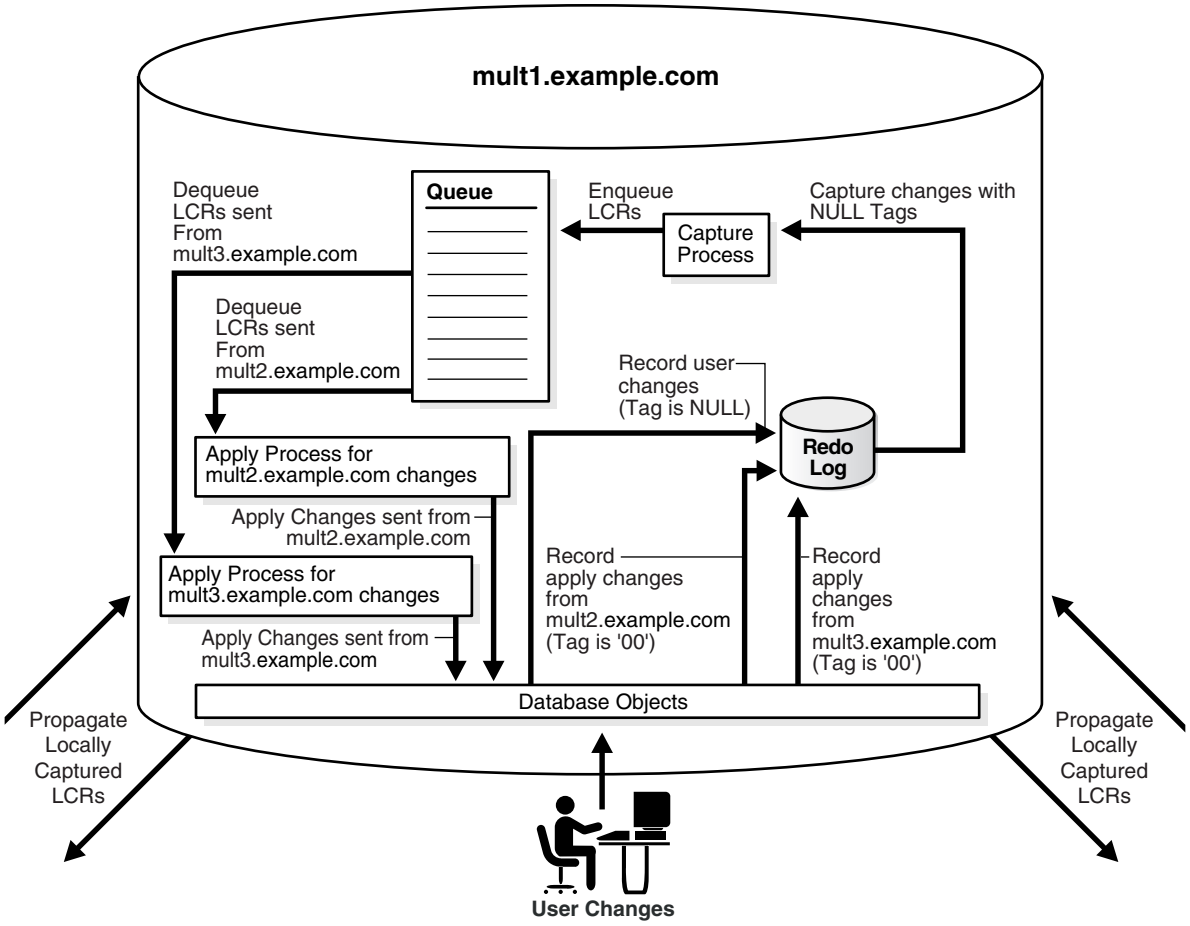
  Each DDL rule should have the following condition:

  ```
  :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 such a condition 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 hrmult schema to every other database. So, in this environment, no changes are lost, and all databases are synchronized. Figure 10–2 illustrates how tags can be used in a database in an n-way replication environment.

Figure 10–2 Tag Use When Each Database Is a Source and Destination Database



> **See Also:** *Oracle Streams Extended Examples* for a detailed illustration of this example

Hub-and-Spoke Replication Environments

A **hub-and-spoke replication** environment is one in which a primary database, or hub, communicates with secondary databases, or spokes. The spokes do not communicate directly with each other. In a hub-and-spoke replication environment, the spokes might or might not allow changes to the replicated database objects.

If the spokes do not allow changes to the replicated database objects, then 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. Change cycling is not possible when none of the secondary databases allow changes to the replicated database objects because changes to the replicated database objects are captured in only one location.

If the spokes allow changes to the replicated database objects, then changes are captured, propagated, and applied in the following way:

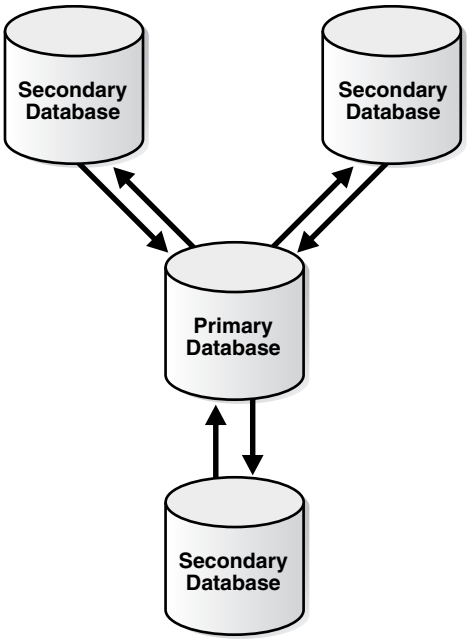
- The primary database captures local changes to the shared data and propagates these changes to all secondary databases, where these changes are applied at each secondary database locally.
- Each secondary database captures local changes to the shared data and propagates these changes to the primary database only, where these changes are applied at the primary database locally.
- The primary database applies changes from each secondary database locally. Next, 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 might 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:** *Oracle Streams Concepts and Administration* 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.example.com` and three secondary databases named `ps2.example.com`, `ps3.example.com`, and `ps4.example.com`. 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. Next, 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 10–3 illustrates a sample environment which has one primary database and multiple secondary databases.

Figure 10–3 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.example.com` 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.example.com` secondary database, then this apply process can 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 if 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.example.com',
    inclusion_rule  => TRUE);
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 process 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 ensure that 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,
    inclusion_rule      => 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 positive 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 ensure that 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 positive 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.example.com`, 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() IS NULL OR :dml.get_tag()!=HEXTORAW('2')
  ```

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

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

  Alternatively, you can add rules to the negative rule set for the propagation so that the propagation discards LCRs with the tag value. 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 `and_condition` parameter in a procedure in the `DBMS_STREAMS_ADM` package to add these conditions to system-created rules, or you can use the `CREATE_RULE` procedure in the `DBMS_RULE_ADM` package to create rules with these conditions. When you specify the condition in the and_condition parameter, specify `:lcr` instead of `:dml` or `:ddl`. See *Oracle Streams Concepts and Administration* for more information about the `and_condition` parameter.

- 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 10–4 illustrates how tags are used at the primary database `ps1.example.com`.

Figure 10–4 Tags Used at the Primary Database

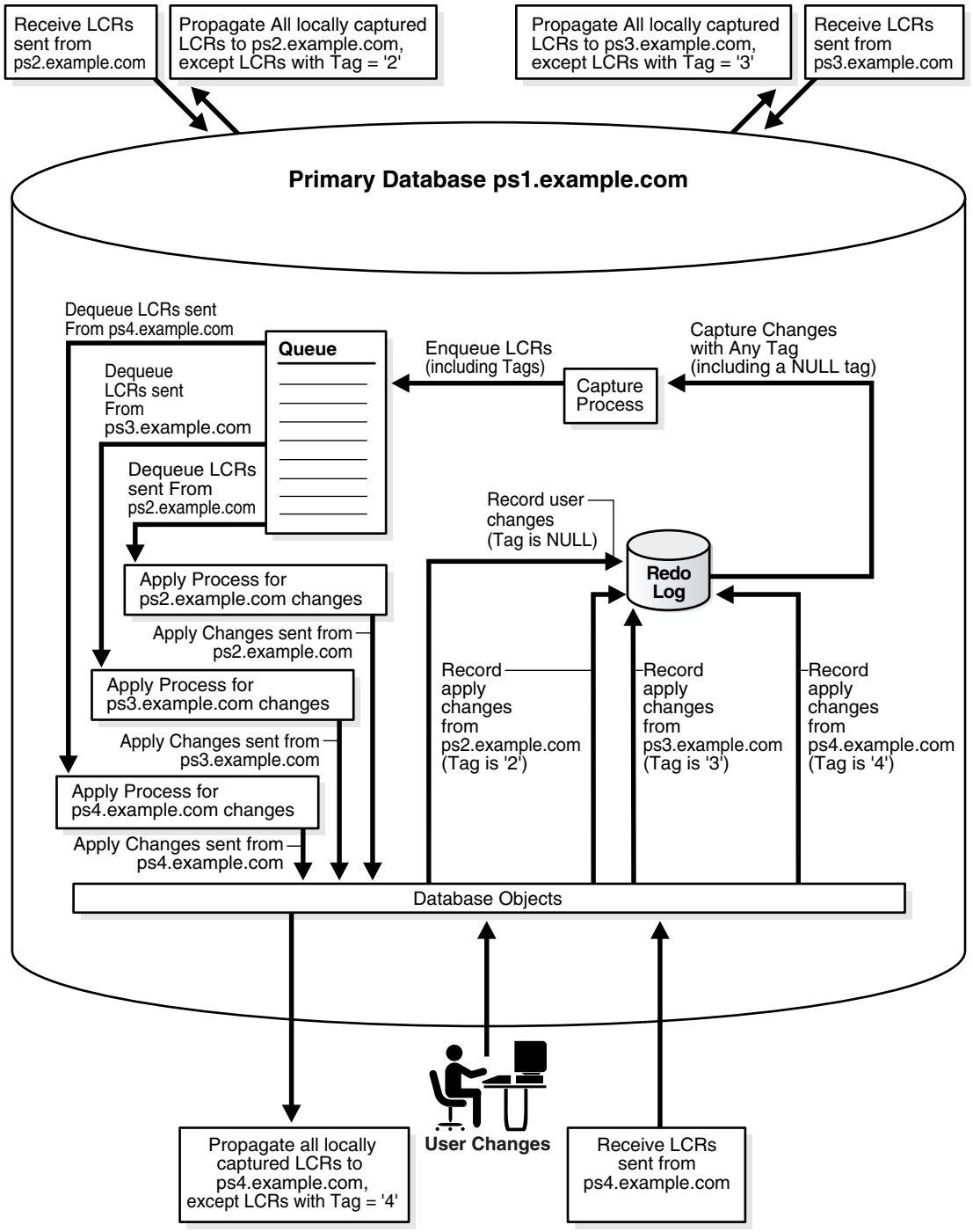
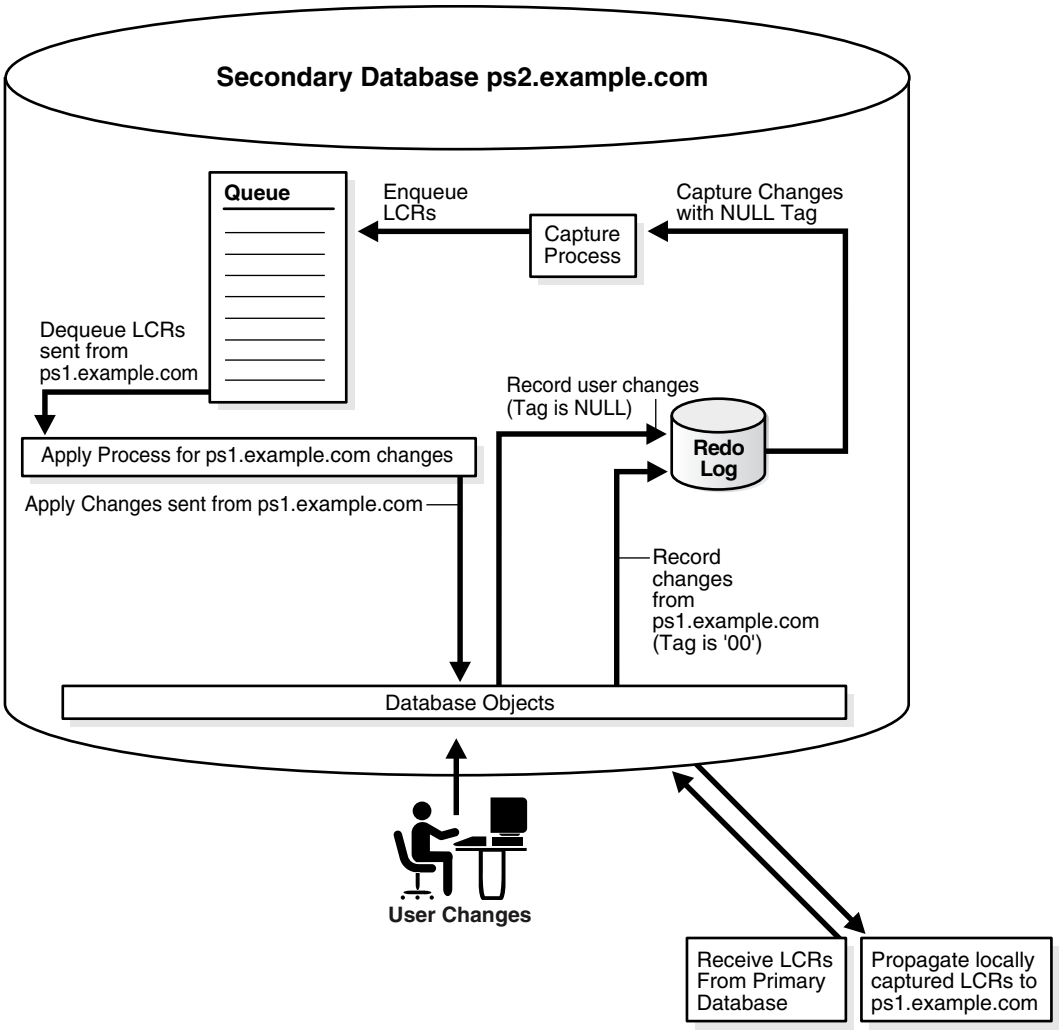


Figure 10–5 illustrates how tags are used at one of the secondary databases (`ps2.example.com`).

Figure 10–5 Tags Used at a Secondary Database



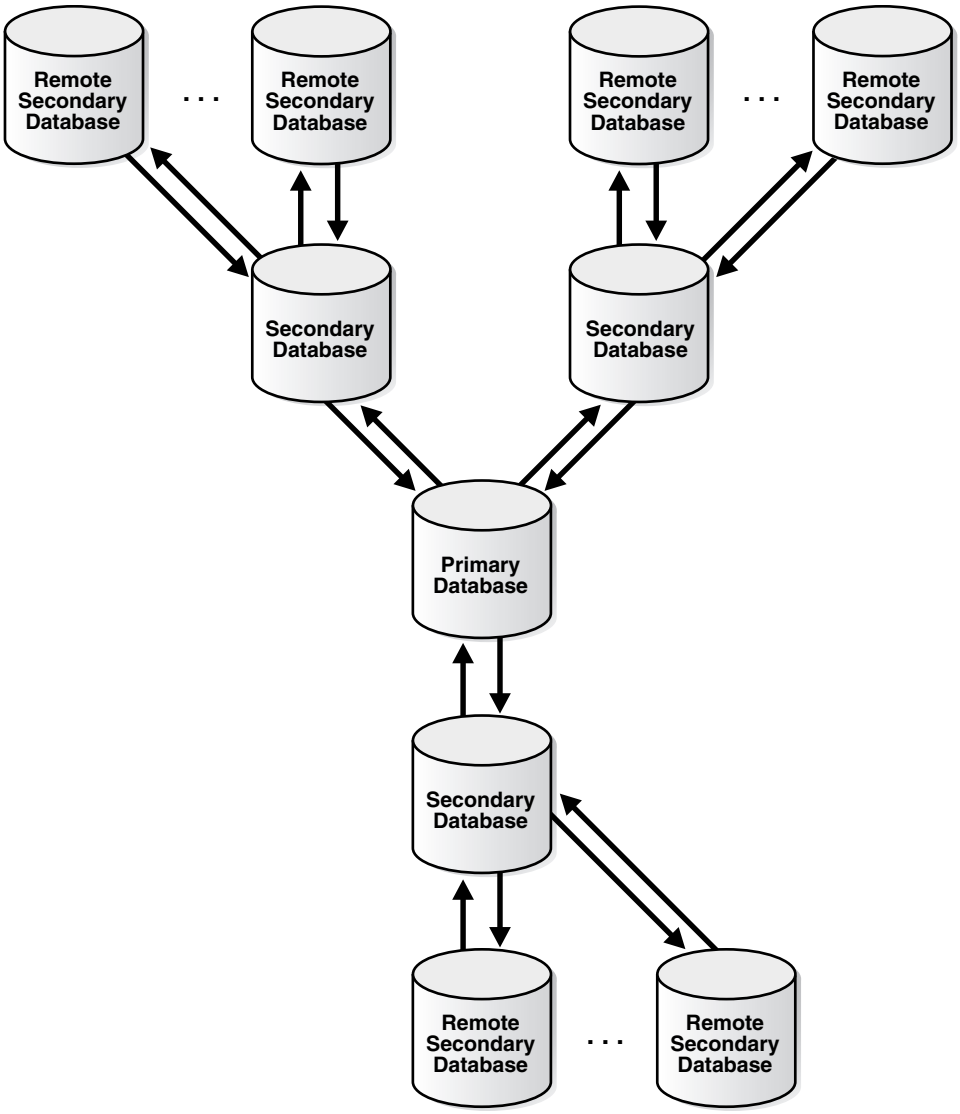
See Also: *Oracle Database 2 Day + Data Replication and Integration Guide* for more information about hub-and-spoke replication environments and for examples that configure such environments

Hub-and-Spoke Replication Environment 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 "Hub-and-Spoke Replication Environments" on page 10-9.

If a remote secondary database allows changes to the replicated database objects, then the remote secondary database does not share data directly with the primary database. Instead, it 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 can be captured and propagated to all of the other databases. Figure 10–6 illustrates an environment with one primary database and multiple extended secondary databases.

Figure 10–6 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 "Hub-and-Spoke Replication Environments" on page 10-9.
- Configure each remote secondary database similar to the way that each secondary database is configured in the example described in "Hub-and-Spoke Replication Environments" on page 10-9. 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 positive 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() IS NULL OR :dml.get_tag()!=HEXTORAW('00')
  ```

  You can use the `and_condition` parameter in a procedure in the `DBMS_STREAMS_ADM` package to add this condition to system-created rules, or you can use the `CREATE_RULE` procedure in the `DBMS_RULE_ADM` package to create rules with this condition. When you specify the condition in the and_condition parameter, specify `:lcr` instead of `:dml` or `:ddl`. See *Oracle Streams Concepts and Administration* for more information about the `and_condition` parameter.

- Configure one propagation from the queue at each secondary database to the queue at each remote secondary database. Each propagation should use a positive 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:

  ```
  :dml.get_tag() IS NULL OR :dml.get_tag()!=HEXTORAW('19')
  ```

  You can use the `and_condition` parameter in a procedure in the `DBMS_STREAMS_ADM` package to add this condition to system-created rules, or you can use the `CREATE_RULE` procedure in the `DBMS_RULE_ADM` package to create rules with this condition. When you specify the condition in the and_condition parameter, specify `:lcr` instead of `:dml` or `:ddl`. See *Oracle Streams Concepts and Administration* for more information about the `and_condition` parameter.

By configuring the environment in this way, you prevent change cycling, and no changes originating at any database are lost.

> **See Also:** *Oracle Database 2 Day + Data Replication and Integration Guide* for more information about hub-and-spoke replication environments and for examples that configure such environments

Managing Oracle 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 Oracle Streams Tags for the Current Session
- Managing Oracle Streams Tags for an Apply Process

> **See Also:** "Monitoring Oracle Streams Tags" on page 10-18

Managing Oracle 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.

The following are considerations for the SET_TAG procedure:

- This procedure is not transactional. That is, the effects of SET_TAG cannot be rolled back.
- If the SET_TAG procedure is run to set a non-NULL session tag before a data dictionary build has been performed on the database, then the redo entries for a transaction that started before the dictionary build might not include the specified tag value for the session. Therefore, perform a data dictionary build before using the SET_TAG procedure in a session. A data dictionary build happens when the DBMS_CAPTURE_ADM.BUILD procedure is run. The BUILD procedure can be run automatically when a capture process is created.

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 Oracle 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 10-5 for conceptual information about how tags are used by an apply process and apply handlers
- *Oracle Streams Concepts and Administration*

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 `strep01_apply` to the hexadecimal value of `'7'`, run the following procedure:

```
BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
     apply_name  =>  'strep01_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 `strep01_apply`.

```
BEGIN
  DBMS_APPLY_ADM.ALTER_APPLY(
    apply_name       => 'strep01_apply',
    remove_apply_tag => TRUE);
END;
/
```

Monitoring Oracle Streams Tags

The following sections contain queries that you can run to display the Oracle Streams tag for the current session and the default tag for each apply process:

- Displaying the Tag Value for the Current Session
- Displaying the Default Tag Value for Each Apply Process

See Also:

- "Managing Oracle Streams Tags" on page 10-16
- *Oracle Database 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 Default Tag Value for Each Apply Process

You can get the default tag for all redo entries generated by each 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 default tag generated in the redo entries by each apply process, run the following query:

```
COLUMN APPLY_NAME HEADING 'Apply Process Name' FORMAT A30
COLUMN APPLY_TAG HEADING 'Tag Value' FORMAT A30

SELECT APPLY_NAME, APPLY_TAG FROM DBA_APPLY;
```

Your output looks similar to the following:

```
Apply Process Name             Tag Value
------------------------------ ------------------------------
APPLY_FROM_MULT2               00
APPLY_FROM_MULT3               00
```

A handler or custom rule-based transformation function associated with an apply process can get the tag by calling the `DBMS_STREAMS.GET_TAG` function.

11

Oracle Streams Heterogeneous Information Sharing

This chapter explains concepts relating to Oracle Streams support for information sharing between Oracle databases and non-Oracle databases.

This chapter contains these topics:

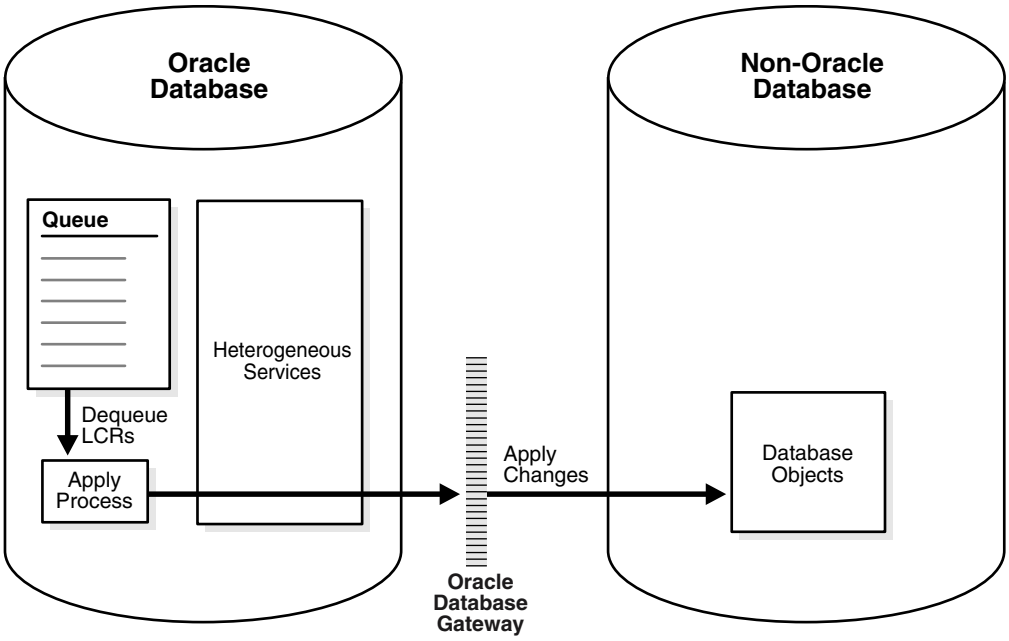
- Oracle to Non-Oracle Data Sharing with Oracle Streams
- Non-Oracle to Oracle Data Sharing with Oracle Streams
- Non-Oracle to Non-Oracle Data Sharing with Oracle Streams

> **Note:** A new feature called XStream is available in Oracle Database 11*g* Release 2 (11.2) and later. XStream enables Oracle Call Interface and Java applications to access the database changes in a stream. See *Oracle Database XStream Guide* for information about XStream.

Oracle to Non-Oracle Data Sharing with Oracle 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 steps that would usually be done at the destination database. That is, the LCRs 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 an Oracle Database Gateway. Figure 11–1 shows an Oracle database sharing data with a non-Oracle database.

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



You should configure the Oracle Database Gateway to use the transaction model `COMMIT_CONFIRM`.

> **See Also:** The Oracle documentation for your specific Oracle Database Gateway for information about using the transaction model `COMMIT_CONFIRM` for your Oracle Database Gateway

Change Capture and Staging in an Oracle to Non-Oracle Environment

In an Oracle to non-Oracle environment, a capture process or a synchronous capture functions the same way as it would in an Oracle-only environment. That is, a capture process finds changes in the redo log, captures them based on its rules, and enqueues the captured changes as logical change records (LCRs) into an `ANYDATA` queue. A synchronous capture uses an internal mechanism to capture changes based on its rules and enqueue the captured changes as row LCRs into an `ANYDATA` queue. In addition, a single capture process or synchronous capture can capture changes that will be applied at both Oracle and non-Oracle databases.

Similarly, the `ANYDATA` queue that stages the 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:**
>
> - *Oracle Streams Concepts and Administration* for general information about capture processes, synchronous captures, staging, and propagations
> - Chapter 1, "Preparing for Oracle Streams Replication" for information about capture processes, synchronous captures, staging, and propagations in an Oracle Streams replication environment

Change Apply in an Oracle to Non-Oracle Environment

An apply process running in an Oracle database uses Heterogeneous Services and an Oracle Database 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 Oracle Streams environment. Instead, the apply process applies the changes directly through a database link to the non-Oracle database.

Note: Oracle Streams apply processes do not support Generic Connectivity.

See Also: *Oracle Streams Concepts and Administration*

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.

Before Creating an Apply Process in an Oracle to Non-Oracle Environment Before you create an apply process that will apply changes to a non-Oracle database, configure Heterogeneous Services, the Oracle Database Gateway, and a database link.

Oracle Streams supports the following Oracle Database Gateways:

- Oracle Database Gateway for Sybase
- Oracle Database Gateway for Informix
- Oracle Database Gateway for SQL Server
- Oracle Database Gateway for DRDA

The database link 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.

See Also:

- *Oracle Database Heterogeneous Connectivity User's Guide* for more information about Heterogeneous Services and Oracle Database Gateway
- The Oracle documentation for your Oracle Database Gateway

Apply Process Creation in an Oracle to Non-Oracle Environment After the database link has been created and is 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:

- *Oracle Database PL/SQL Packages and Types Reference* for more information about the procedures in the `DBMS_APPLY_ADM` package
- *Oracle Streams Concepts and Administration* 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:** *Oracle Streams Concepts and Administration*

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 can use multiple apply processes to apply changes a non-Oracle database.

Procedure DML Handlers in an Oracle to Non-Oracle Heterogeneous Environment If you use a procedure 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:** *Oracle Streams Concepts and Administration* for information about message processing options for an apply process

Message Handlers in an Oracle to Non-Oracle Heterogeneous Environment If you want to use a message handler to process user 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:** *Oracle Streams Concepts and Administration* for information about message processing options and managing message handlers

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 the error queue in the Oracle database.

Data Types 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 data types:

- `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 data types to non-Oracle databases: CLOB, NCLOB, BLOB, BFILE, LONG, LONG RAW, ROWID, UROWID, user-defined types (including object types, REFs, varrays, and nested tables), and Oracle-supplied types (including Any types, XML types, spatial types, and media types). The apply process raises an error when an LCR contains a data type that is not listed, and the transaction containing the LCR that caused the error is moved to the error queue in the Oracle database.

Each Oracle Database Gateway might have further limitations regarding data types. For a data type to be supported in an Oracle to non-Oracle environment, the data type must be supported by both Oracle Streams and the Oracle Database Gateway being used.

See Also:

- *Oracle Database SQL Language Reference* for more information about these data types
- The Oracle documentation for your specific Oracle Database Gateway

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.

   The following is an example that uses the DBMS_HS_PASSTHROUGH package to create the hr.regions table in the het.example.com non-Oracle database:

```
DECLARE
 ret INTEGER;
BEGIN
ret := DBMS_HS_PASSTHROUGH.EXECUTE_IMMEDIATE@het.example.com (
  'CREATE TABLE regions (region_id INTEGER, region_name VARCHAR(50))');
END;
/
COMMIT;
```

See Also: *Oracle Database Heterogeneous Connectivity User's Guide* and the Oracle documentation for your specific Oracle Database Gateway for more information about Heterogeneous Services and Oracle Database Gateway

2. If the changes that will be shared between the Oracle and non-Oracle database are captured by a capture process or synchronous capture 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 8-8

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.example.com` 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.example.com 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.

If the Oracle Database Gateway you are using supports the Heterogeneous Services callback functionality, then you can replace the loop in the previous example with the following SQL statement:

```
INSERT INTO hr.region@het.example.com SELECT * FROM hr.region@!;
```

Note: The user who creates and runs the procedure in the previous example must have `EXECUTE` privilege on the `DBMS_FLASHBACK` package and all privileges on the tables involved.

See Also: *Oracle Database Heterogeneous Connectivity User's Guide* and the Oracle documentation for your specific Oracle Database Gateway for information about callback functionality and your Oracle Database Gateway

4. Set the instantiation SCN for the table at the non-Oracle database. 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. Ensure that 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 8-36 and *Oracle Database PL/SQL Packages and Types Reference* for more information about the `SET_TABLE_INSTANTIATION_SCN` procedure

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: *Oracle Streams Concepts and Administration* for more information about rule-based transformations

Messaging Gateway and Oracle 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:** *Oracle Streams Advanced Queuing User's Guide* 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 the error 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.

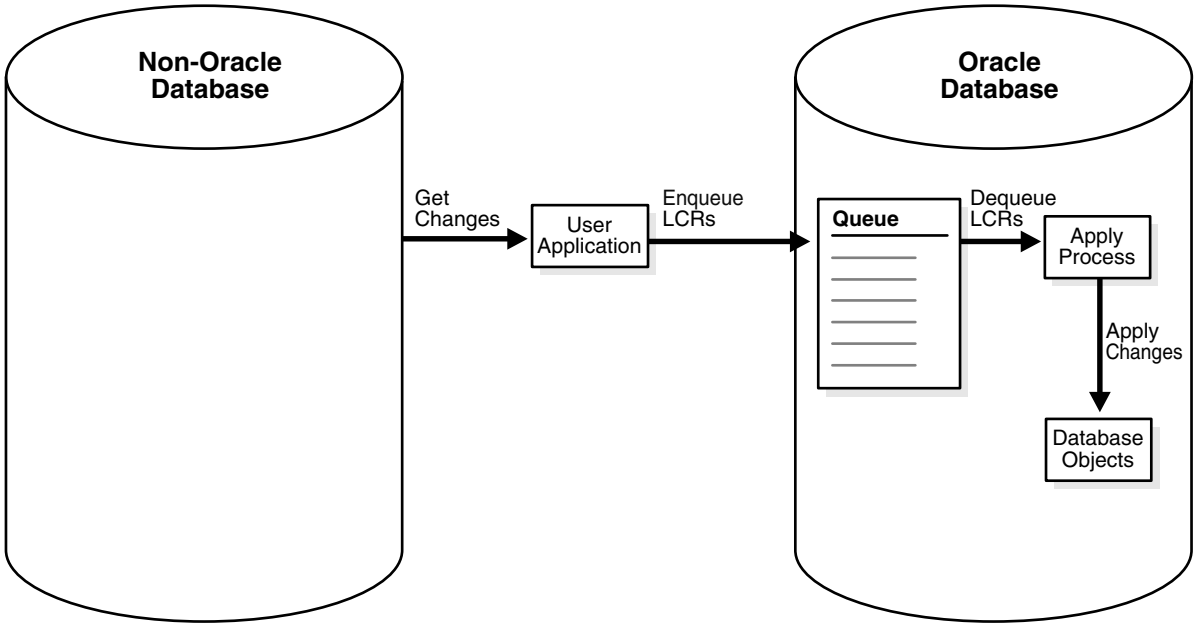
Example Oracle to Non-Oracle Streams Environment

Oracle Streams Extended Examples contains a detailed example that includes sharing data in an Oracle to non-Oracle Streams environment.

Non-Oracle to Oracle Data Sharing with Oracle 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). Next, the application must enqueue the LCRs in an Oracle database using the `DBMS_STREAMS_MESSAGING` package or the `DBMS_AQ` package. The application must commit after enqueuing all LCRs in each transaction. Figure 11–2 shows a non-Oracle databases sharing data with an Oracle database.

Figure 11–2 Non-Oracle to Oracle Heterogeneous Data Sharing



Change Capture 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 enqueuing the LCRs at the Oracle database, the application is completely responsible for change capture. Therefore, 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 can enqueue multiple transactions concurrently, but the transactions must be committed in the same order as the transactions on the non-Oracle source database.

> **See Also:** "Constructing and Enqueuing LCRs" on page 14-2 for more information about constructing and enqueuing LCRs

Staging in a Non-Oracle to Oracle Environment

To ensure the same transactional consistency at both the Oracle database where changes are applied and the non-Oracle database where changes originate, 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.

Also, the queue at the Oracle database should be a commit-time queue. A commit-time queue orders LCRs by approximate commit system change number (approximate CSCN) of the transaction that includes the LCRs. Commit-time queues preserve transactional dependency ordering between LCRs in the queue, if the application that enqueued the LCRs commits the transactions in the correct order. Also, commit-time queues ensure consistent browses of LCRs in a queue.

> **See Also:** *Oracle Streams Concepts and Administration* for more information about transactional queues and commit-time queues

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 LCR from its associated queue based on apply process rules, performs any rule-based transformation, and either sends the LCR 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.

The apply process should be configured to apply persistent LCRs, not captured LCRs. So, the apply process should be created using the `CREATE_APPLY` procedure in the `DBMS_APPLY_ADM` package, and the `apply_captured` parameter should be set to `FALSE` when you run this procedure. After the apply process is created, you can use procedures in the `DBMS_STREAMS_ADM` package to add rules for LCRs to the apply process rule sets.

See Also:

- *Oracle Streams Concepts and Administration* for more information about apply processes, rules, and rule-based transformations
- Chapter 9, "Oracle Streams Conflict Resolution"

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: *Oracle Database Utilities* for information about using SQL*Loader

Non-Oracle to Non-Oracle Data Sharing with Oracle Streams

Oracle 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 Oracle Streams in an Oracle database as an intermediate database between two non-Oracle databases.

For example, a non-Oracle to non-Oracle environment can consist of the following databases:

- A non-Oracle database named `het1.example.com`
- An Oracle database named `dbs1.example.com`
- A non-Oracle database named `het2.example.com`

A user application assembles changes at `het1.example.com` and enqueues them in `dbs1.example.com`. Next, the apply process at `dbs1.example.com` applies the changes to `het2.example.com` using Heterogeneous Services and an Oracle Database Gateway. Another apply process at `dbs1.example.com` could apply some or all of the changes in the queue locally at `dbs1.example.com`. One or more propagations at `dbs1.example.com` could propagate some or all of the changes in the queue to other Oracle databases.

Part II

Administering Oracle Streams Replication

This part describes managing Oracle Streams replication and contains the following chapters:

- Chapter 12, "Managing Oracle Streams Replication"
- Chapter 13, "Comparing and Converging Data"
- Chapter 14, "Managing Logical Change Records (LCRs)"

12

Managing Oracle Streams Replication

This chapter contains instructions for managing an Oracle Streams replication environment.

This chapter contains these topics:

- About Managing Oracle Streams
- Tracking LCRs Through a Stream
- Splitting and Merging an Oracle Streams Destination
- Changing the DBID or Global Name of a Source Database
- Resynchronizing a Source Database in a Multiple-Source Environment
- Performing Database Point-in-Time Recovery in an Oracle Streams Environment
- Running Flashback Queries in an Oracle Streams Replication Environment
- Recovering from Operation Errors

About Managing Oracle Streams

After an Oracle Streams replication environment is in place, you can manage the Oracle Streams components at each database. Management includes administering the components. For example, you can set capture process parameters to modify the behavior of a capture process. Management also includes monitoring the Oracle Streams components and troubleshooting them if there are problems.

The following documentation provides instructions for managing Oracle Streams:

- *Oracle Streams Concepts and Administration* provides detailed instructions about managing Oracle Streams components.
- *Oracle Database 2 Day + Data Replication and Integration Guide* provides instructions about performing the most common management tasks.
- *Oracle Streams Replication Administrator's Guide* (this document) provides instructions that are specific to an Oracle Streams replication environment.
- The online help for the Oracle Streams interface in Oracle Enterprise Manager provides information about managing Oracle Streams with Oracle Enterprise Manager.

Tracking LCRs Through a Stream

A logical change record (LCR) typically flows through a stream in the following way:

1. A database change is captured, formatted into an LCR, and enqueued. A capture process or a synchronous capture can capture database changes implicitly. An application or user can construct and enqueue LCRs to capture database changes explicitly.
2. One or more propagations send the LCR to other databases in the Oracle Streams environment.
3. One or more apply processes dequeue the LCR and process it.

You can track an LCR through a stream using one of the following methods:

- When LCRs are captured by a capture process, you can set the `message_tracking_frequency` capture process parameter to `1` or another relatively low value.
- When LCRs are captured by a capture process or a synchronous capture, or when LCRs are constructed by an application, you can run the `SET_MESSAGE_TRACKING` procedure in the `DBMS_STREAMS_ADM` package.

LCR tracking is useful if LCRs are not being applied as expected by one or more apply processes. When this happens, you can use LCR tracking to determine where the LCRs are stopping in the stream and address the problem at that location.

After using one of these methods to track LCRs, use the `V$STREAMS_MESSAGE_TRACKING` view to monitor the progress of LCRs through a stream. By tracking an LCR through the stream, you can determine where the LCR is blocked. After LCR tracking is started, each LCR includes a tracking label.

When LCR tracking is started using the `message_tracking_frequency` capture process parameter, the tracking label is *`capture_process_name`*`:AUTOTRACK`, where *`capture_process_name`* is the name of the capture process. Only the first 20 bytes of the capture process name are used; the rest is truncated if it exceeds 20 bytes.

The `SET_MESSAGE_TRACKING` procedure enables you to specify a tracking label that becomes part of each LCR generated by the current session. Using this tracking label, you can query the `V$STREAMS_MESSAGE_TRACKING` view to track the LCRs through the stream and see how they were processed by each Oracle Streams client. When you use the `SET_MESSAGE_TRACKING` procedure, the following LCRs are tracked:

- When a capture process or a synchronous capture captures an LCR, and a tracking label is set for the session that made the captured database change, the tracking label is included in the LCR automatically.
- When a user or application constructs an LCR and a tracking label is set for the session that constructs the LCR, the tracking label is included in the LCR automatically.

To track LCRs through a stream, complete the following steps:

1. Start LCR tracking.

   You can start LCR tracking in one of the following ways:

   - Connect to database running the capture process and set the `message_tracking_frequency` capture process parameter to `1` or another relatively low value. After setting the capture process parameter, proceed to Step 2.

     See *Oracle Database 2 Day + Data Replication and Integration Guide* or *Oracle Streams Concepts and Administration* for information about setting capture process parameters.

     See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.

- Run the SET_MESSAGE_TRACKING procedure in the DBMS_STREAMS_ADM package by completing the following steps:

a. In SQL*Plus, start a session. To use a tracking label for database changes captured by a capture process or synchronous capture, connect to the source database for the capture process or synchronous capture.

b. Begin message tracking:

```
BEGIN
  DBMS_STREAMS_ADM.SET_MESSAGE_TRACKING(
    tracking_label => 'TRACK_LCRS');
END;
/
```

You can use any label you choose to track LCRs. This example uses the TRACK_LCRS label.

Information about the LCRs is tracked in memory, and the V$STREAMS_MESSAGE_TRACKING dynamic performance view is populated with information about the LCRs.

c. Optionally, to ensure that message tracking is set in the session, query the tracking label:

```
SELECT DBMS_STREAMS_ADM.GET_MESSAGE_TRACKING() TRACKING_LABEL FROM DUAL;
```

This query should return the tracking label you specified in Step b:

```
TRACKING_LABEL
------------------------------------------------------------------------
TRACK_LCRS
```

2. Make changes to the source database that will be captured by the capture process or synchronous capture that starts the stream, or construct and enqueue the LCRs you want to track. Typically, these LCRs are for testing purposes only. For example, you can insert several dummy rows into a table and then modify these rows. When the testing is complete, you can delete the rows.

3. Monitor the entire Oracle Streams environment to track the LCRs. To do so, query the V$STREAMS_MESSAGE_TRACKING view at each database that processes the LCRs.

For example, run the following query at each database:

```
COLUMN COMPONENT_NAME HEADING 'Component|Name' FORMAT A10
COLUMN COMPONENT_TYPE HEADING 'Component|Type' FORMAT A12
COLUMN ACTION HEADING 'Action' FORMAT A11
COLUMN SOURCE_DATABASE_NAME HEADING 'Source|Database' FORMAT A10
COLUMN OBJECT_OWNER HEADING 'Object|Owner' FORMAT A6
COLUMN OBJECT_NAME HEADING 'Object|Name' FORMAT A10
COLUMN COMMAND_TYPE HEADING 'Command|Type' FORMAT A7

SELECT COMPONENT_NAME,
       COMPONENT_TYPE,
       ACTION,
       SOURCE_DATABASE_NAME,
       OBJECT_OWNER,
       OBJECT_NAME,
       COMMAND_TYPE
   FROM V$STREAMS_MESSAGE_TRACKING;
```

Ensure that you specify the correct tracking label in the WHERE clause.

These queries will show how the LCRs were processed at each database. If the LCRs are not being applied at destination databases, then these queries will show where in the stream the LCRs are stopping.

For example, the output at a source database with a synchronous capture is similar to the following:

```
Component  Component                Source     Object Object     Command
Name       Type         Action      Database   Owner  Name       Type
---------- ------------ ----------- ---------- ------ ---------- -------
CAPTURE    SYNCHRONOUS  Create      HUB.EXAMPL HR     EMPLOYEES  UPDATE
           CAPTURE                  E.COM
CAPTURE    SYNCHRONOUS  Rule evalua HUB.EXAMPL HR     EMPLOYEES  UPDATE
           CAPTURE      tion        E.COM
CAPTURE    SYNCHRONOUS  Enqueue     HUB.EXAMPL HR     EMPLOYEES  UPDATE
           CAPTURE                  E.COM
```

The output at a destination database with an apply process is similar to the following:

```
Component  Component                Source     Object Object     Command
Name       Type         Action      Database   Owner  Name       Type
---------- ------------ ----------- ---------- ------ ---------- -------
APPLY_SYNC APPLY READER Dequeue     HUB.EXAMPL HR     EMPLOYEES  UPDATE
_CAP                                E.COM
APPLY_SYNC APPLY READER Dequeue     HUB.EXAMPL HR     EMPLOYEES  UPDATE
_CAP                                E.COM
APPLY_SYNC APPLY READER Dequeue     HUB.EXAMPL HR     EMPLOYEES  UPDATE
_CAP                                E.COM
```

You can query additional columns in the `V$STREAMS_MESSAGE_TRACKING` view to display more information. For example, the `ACTION_DETAILS` column provides detailed information about each action.

4. Stop message tracking. Complete one of the following actions based your choice in Step 1:

   - If you set the `message_tracking_frequency` capture process parameter in Step 1, then set this parameter back to its default value. The default is to track every two-millionth message.

     To set this capture process parameter back to its default value, connect to database running the capture process and set the `message_tracking_frequency` capture process parameter to `NULL`.

     See *Oracle Database 2 Day + Data Replication and Integration Guide* or *Oracle Streams Concepts and Administration* for information about setting capture process parameters.

   - If you started message tracking in the current session, then stop message tracking in the session.

     To stop message tracking in the current session, set the `tracking_label` parameter to `NULL` in the `SET_MESSAGE_TRACKING` procedure:

     ```
     BEGIN
       DBMS_STREAMS_ADM.SET_MESSAGE_TRACKING(
         tracking_label => NULL,
         actions        => DBMS_STREAMS_ADM.ACTION_MEMORY);
     END;
     /
     ```

See Also: *Oracle Database PL/SQL Packages and Types Reference* for information about the `message_tracking_frequency` capture process parameter

Splitting and Merging an Oracle Streams Destination

The following sections describe how to split and merge streams and provide examples that do so:

- About Splitting and Merging Oracle Streams
- Split and Merge Options
- Examples That Split and Merge Oracle Streams

About Splitting and Merging Oracle Streams

Splitting and merging an Oracle Streams destination is useful under the following conditions:

- A single capture process captures changes that are sent to two or more apply processes.
- An apply process stops accepting changes captured by the capture process. The apply process might stop accepting changes if, for example, the apply process is disabled, the database that contains the apply process goes down, there is a network problem, the computer system running the database that contains the apply process goes down, or for some other reason.

When these conditions are met, it is best to split the problem destination off from the other destinations. The reason to split the destination off depends on whether the configuration uses the combined capture and apply optimization:

- If the apply process at the problem destination is part of a combined capture and apply optimization and the destination is not split off, then performance will suffer when the destination becomes available again. In this case, the capture process must capture the changes that must now be applied at the destination previously split off. The other destinations will not receive more recent changes until the problem destination has caught up. However, if the problem destination is split off, then it can catch up to the other destinations independently, without affecting the other destinations.
- If the apply process at the destination is not part of a combined capture and apply optimization, then captured changes that cannot be sent to the problem destination queue remain in the source queue, causing the source queue size to increase. Eventually, the source queue will spill captured logical change records (LCRs) to hard disk, and the performance of the Oracle Streams replication environment will suffer.

Split and merge operations are possible in the following types of Oracle Streams replication environments:

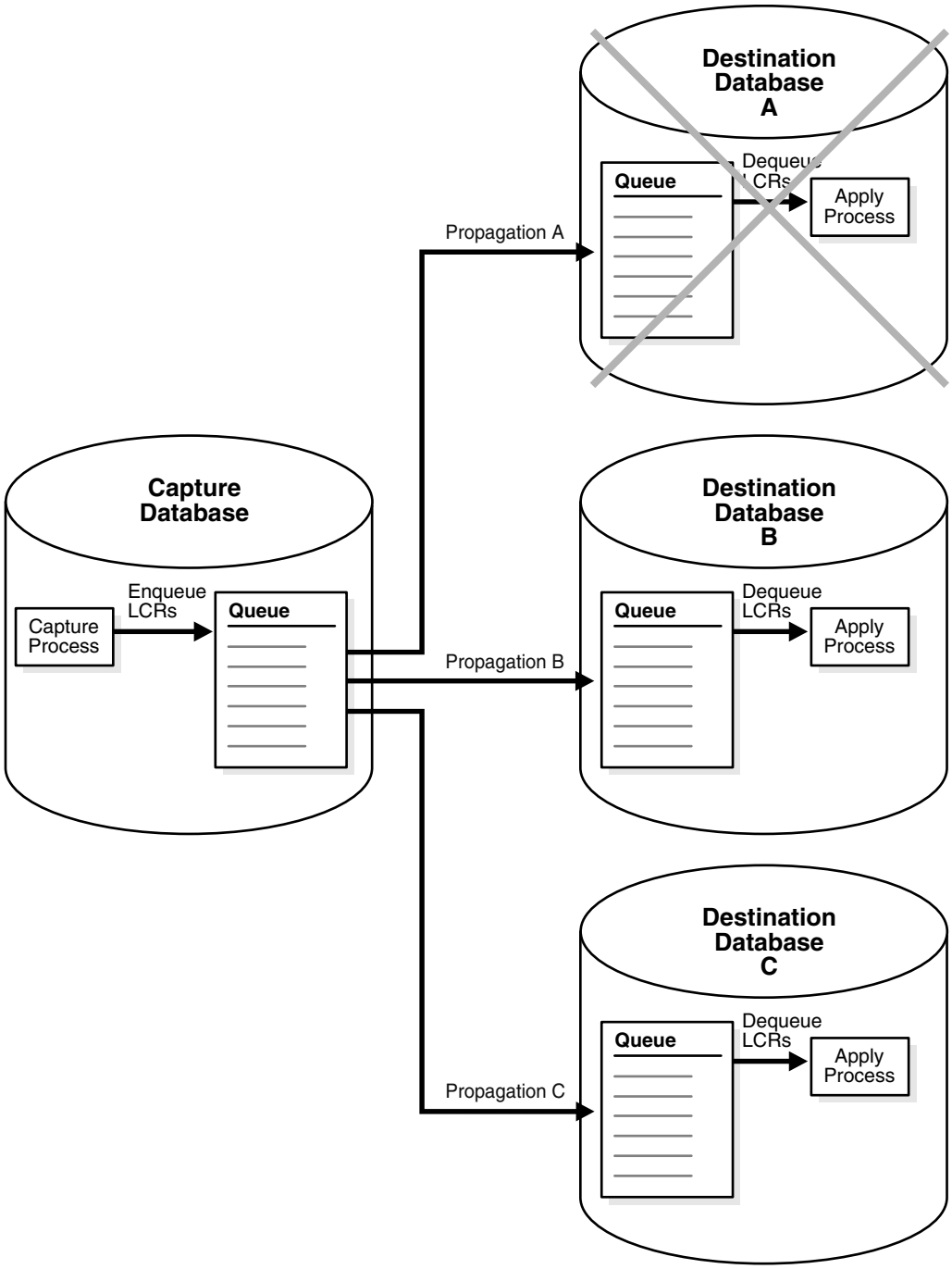
- Changes captured by a single capture process are sent to multiple remote destinations using propagations and are applied by apply processes at the remote destinations.
- Changes captured by a single capture process are applied locally by multiple apply processes on the same database that is running the capture process.

- Changes captured by a single capture process are sent to one or more remote destinations using propagations and are applied locally by one or more apply processes on the same database that is running the capture process.

For environment with local capture and apply, split and merge operations are possible when the capture process and apply processes share the same queue, and when a propagation sends changes from the capture process's queue to an apply process's queue within the one database.

Figure 12–1 shows an Oracle Streams replication environment that uses propagations to send changes to multiple destinations. In this example, destination database A is down.

Figure 12–1 Problem Destination in an Oracle Streams Replication Environment



You can use the following data dictionary views to determine when there is a problem with a stream:

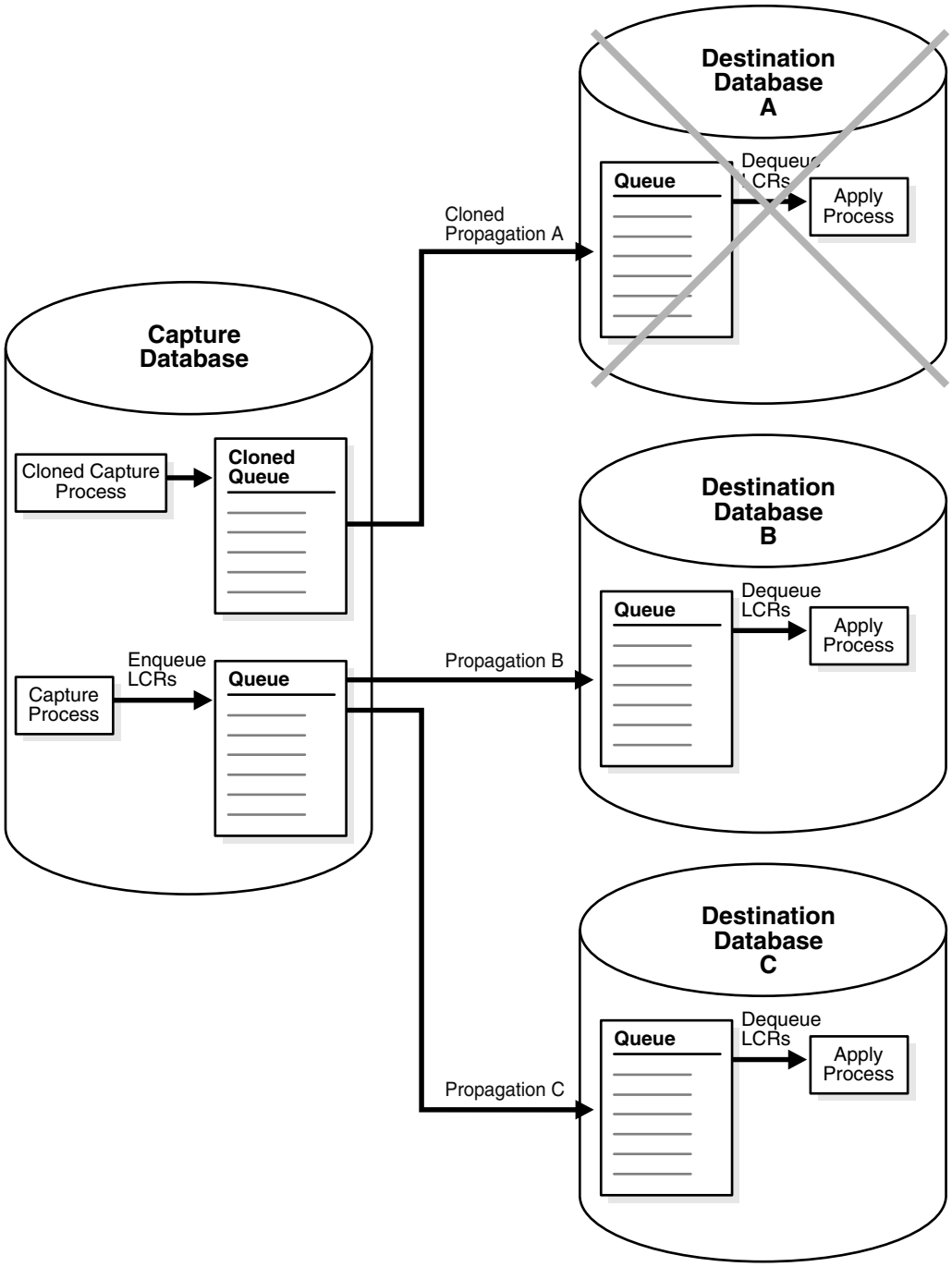
- Query the `V$BUFFERED_QUEUES` view to identify how many messages are in a buffered queue and how many of these messages have spilled to hard disk.
- When propagations are used, query the `DBA_PROPAGATION` and `V$PROPAGATION_SENDER` views to show the propagations in a database and the status of each propagation

To avoid degraded performance in this situation, split the stream that flows to the problem database off from the other streams flowing from the capture process. When the problem is corrected, merge the stream back into the other streams flowing from the capture process.

You can configure capture process parameters to split and merge a problem stream automatically, or you can split and merge a problem stream manually. Either way, the `SPLIT_STREAMS`, `MERGE_STREAMS_JOB`, and `MERGE_STREAMS` procedures in the `DBMS_STREAMS_ADM` package are used. The `SPLIT_STREAMS` procedure splits off the stream for the problem destination from all of the other streams flowing from a capture process to other destinations. The `SPLIT_STREAMS` procedure always clones the capture process and the queue. The `SPLIT_STREAMS` procedure also clones the propagation in an environment that sends changes to remote destination databases. The cloned versions of these components are used by the stream that is split off. While the problem stream is split off, the streams to other destinations proceed as usual.

Figure 12–2 shows the cloned stream created by the `SPLIT_STREAMS` procedure.

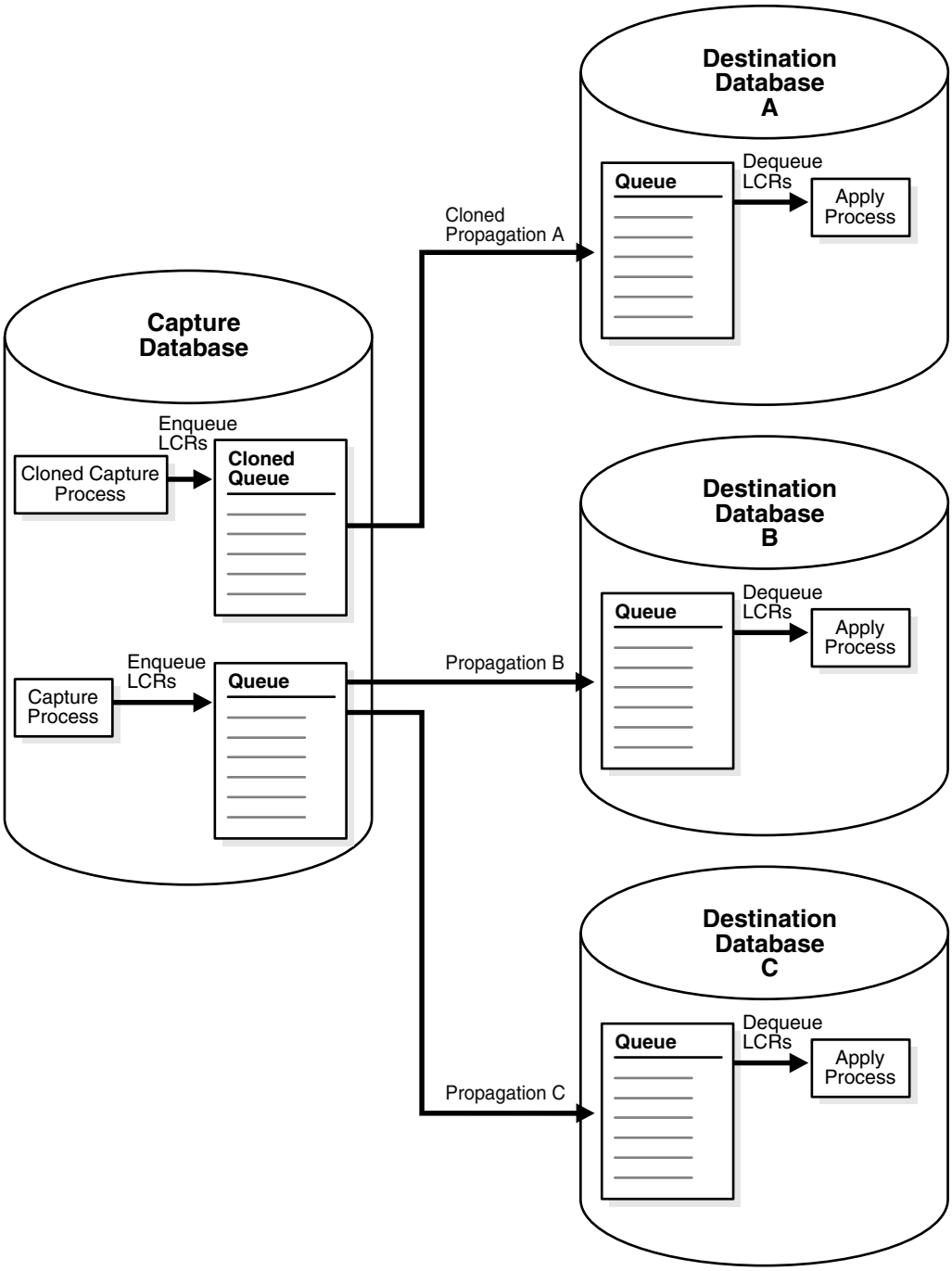
Figure 12–2 Splitting Oracle Streams



When the problem destination becomes available again, the cloned stream begins to send captured changes to the destination database again.

Figure 12–3 shows a destination database A that is up and running and a cloned capture process that is enabled at the capture database. The cloned stream begins to flow and starts to catch up to the original streams.

Figure 12–3 Cloned Stream Begins Flowing and Starts to Catch Up to One Original Stream



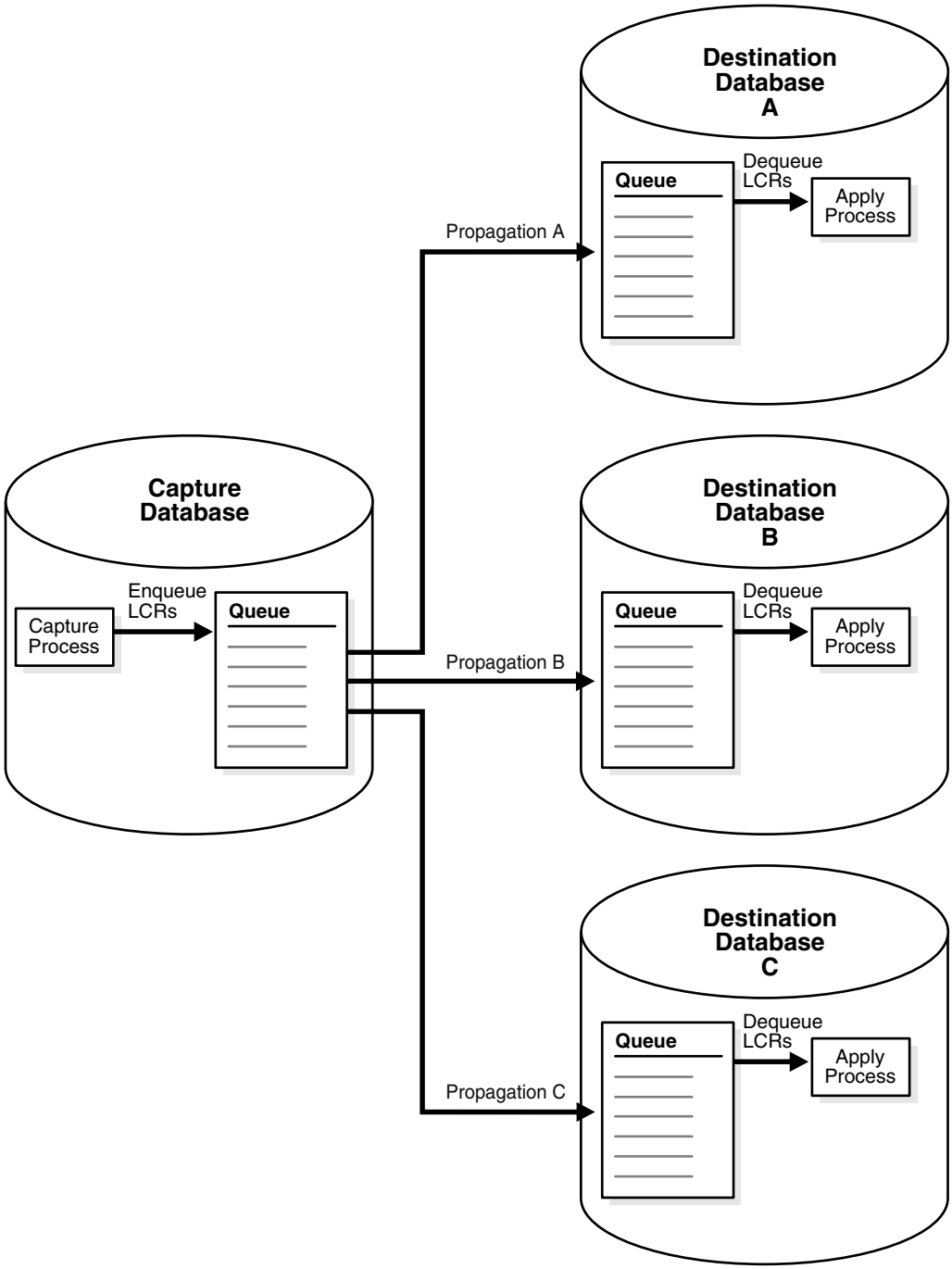
When the cloned stream catches up to one of the original streams, one of the following procedures merges the streams:

- The `MERGE_STREAMS` procedure merges the stream that was split off back into the other streams flowing from the original capture process.
- The `MERGE_STREAMS_JOB` procedure determines whether the streams are within the user-specified merge threshold. If they are, then the `MERGE_STREAMS_JOB` procedure runs the `MERGE_STREAMS` procedure. If the streams are not within the merge threshold, then the `MERGE_STREAMS_JOB` procedure does nothing.

Typically, it is best to run the `MERGE_STREAMS_JOB` procedure instead of running the `MERGE_STREAMS` procedure directly, because the `MERGE_STREAMS_JOB` procedure automatically determines whether the streams are ready to merge before merging them.

Figure 12–4 shows the results of running the `MERGE_STREAMS` procedure. The Oracle Streams replication environment has its original components, and all of the streams are flowing normally.

Figure 12–4 Merging Oracle Streams



> **See Also:** *Oracle Streams Concepts and Administration* for information about combined capture and apply

Split and Merge Options

The following split and merge options are available:

- Automatic Split and Merge
- Manual Split and Automatic Merge
- Manual Split and Merge With Generated Scripts

Automatic Split and Merge

You can set two capture process parameters, `split_threshold` and `merge_theshold`, so that Oracle Streams performs split and merge operations automatically. When these parameters are set to specify automatic split and merge, an Oracle Scheduler job monitors the streams flowing from the capture process. When an Oracle Scheduler job identifies a problem with a stream, the job splits the problem stream off from the other streams flowing from the capture process. When a split operation is complete, a new Oracle Scheduler merge job monitors the split stream. When the problem is corrected, this job merges the stream back with the other streams.

When the `split_threshold` capture process parameter is set to `INFINITE`, automatic splitting is disabled. When the `split_threshold` parameter is not set to `INFINITE`, automatic splitting is enabled. Automatic splitting only occurs when communication with an apply process has been lost for the number of seconds specified in the `split_threshold` parameter. For example, communication with an apply process is lost when an apply process becomes disabled or a destination database goes down. Automatic splitting does not occur when one stream is processing changes slower than other streams.

When a stream is split, a cloned capture process is created. The cloned capture process might be enabled or disabled after the split depending on whether the configuration uses the combined capture and apply optimization:

- If the apply process is part of a combined capture and apply optimization, then the cloned capture process is enabled. The cloned capture process does not capture any changes until the apply process is enabled and communication is established with the apply process.
- If the apply process is not part of a combined capture and apply optimization, then the cloned capture process is disabled so that LCRs do not build up in a queue. When the apply process is enabled and the cloned stream can flow, you can start the cloned capture process manually.

The split stream is merged back with the original streams automatically when the difference, in seconds, between `CAPTURE_MESSAGE_CREATE_TIME` in the `GV$STREAMS_CAPTURE` view of the cloned capture process and the original capture process is less than or equal to the value specified for the `merge_threshold` capture process parameter. The `CAPTURE_MESSAGE_CREATE_TIME` records the time when a captured change was recorded in the redo log. If the difference is greater than the value specified by this capture process parameter, then automatic merge does not begin, and the value is recorded in the `LAG` column of the `DBA_STREAMS_SPLIT_MERGE` view.

When the capture process and the apply process for a stream run in different database instances, automatic split and merge is always possible for the stream. When a capture

process and apply process for a stream run on the same database instance, automatic split and merge is possible only when all of the following conditions are met:

- The capture process and apply process use the same queue.
- The apply process has no errors in its error queue.
- The apply process is not an XStream outbound server.
- The apply process is stopped.
- No messages have spilled from the buffered queue to the hard disk.

See Also:

- "Splitting and Merging an Oracle Streams Destination Automatically" on page 12-15 for an example
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the capture process parameters
- *Oracle Streams Concepts and Administration* for information about monitoring automatic split and merge operations
- *Oracle Database XStream Guide*

Manual Split and Automatic Merge

When you split streams manually with the `SPLIT_STREAMS` procedure, the `auto_merge_threshold` procedure parameter gives you the option of automatically merging the stream back to the original capture process when the problem at the destination is corrected. After the apply process for the problem stream is accepting changes, you can start the cloned capture process and wait for the cloned capture process to catch up to the original capture process. When the cloned capture process nearly catches up, the `auto_merge_threshold` parameter setting determines whether the split stream is merged automatically or manually:

- When `auto_merge_threshold` is set to a positive number, the `SPLIT_STREAMS` procedure creates an Oracle Scheduler job with a schedule. The job runs the `MERGE_STREAMS_JOB` procedure and specifies a merge threshold equal to the value specified in the `auto_merge_threshold` parameter. You can modify the schedule for a job after it is created.

  In this case, the split stream is merged back with the original streams automatically when the difference, in seconds, between `CAPTURE_MESSAGE_CREATE_TIME` in the `GV$STREAMS_CAPTURE` view of the cloned capture process and the original capture process is less than or equal to the value specified for the `auto_merge_threshold` parameter. The `CAPTURE_MESSAGE_CREATE_TIME` records the time when a captured change was recorded in the redo log.
- When `auto_merge_threshold` is set to `NULL` or `0` (zero), the split stream is not merged back with the original streams automatically. To merge the split stream with the original streams, run the `MERGE_STREAMS_JOB` or `MERGE_STREAMS` procedure manually.

See Also: "Splitting an Oracle Streams Destination Manually and Merging It Automatically" on page 12-17 for an example

Manual Split and Merge With Generated Scripts

The `SPLIT_STREAMS` and `MERGE_STREAMS` procedures can perform actions directly or generate a script that performs the actions when the script is run. Using a procedure to perform actions directly is simpler than running a script, and the split or merge

operation is performed immediately. However, you might choose to generate a script for the following reasons:

- You want to review the actions performed by the procedure before splitting or merging streams.
- You want to modify the script to customize its actions.

For example, you might choose to modify the script if you want to change the rules in the rule set for the cloned capture process. In some Oracle Streams replication environments, only a subset of the changes made to the source database are sent to each destination database, and each destination database might receive a different subset of the changes. In such an environment, you can modify the rule set for the cloned capture process so that it only captures changes that are propagated by the cloned propagation.

The `perform_actions` parameter in each procedure controls whether the procedure performs actions directly:

- To split or merge streams directly when you run one of these procedures, set the `perform_actions` parameter to `TRUE`. The default value for this parameter is `TRUE`.
- To generate a script when you run one of these procedures, set the `perform_actions` parameter to `FALSE`, and use the `script_name` and `script_directory_object` parameters to specify the name and location of the script.

> **See Also:** "Splitting and Merging an Oracle Streams Destination Manually With Scripts" on page 12-20 for an example

Examples That Split and Merge Oracle Streams

The following sections provide instructions for splitting and merging streams:

- Splitting and Merging an Oracle Streams Destination Automatically
- Splitting an Oracle Streams Destination Manually and Merging It Automatically
- Splitting and Merging an Oracle Streams Destination Manually With Scripts

These examples make the following assumptions about the Oracle Streams replication environment:

- A single capture process named `strms_capture` captures changes that are sent to three destination databases.
- The propagations that send these changes to the destination queues at the destination databases are the following:
  - `strms_prop_a`
  - `strms_prop_b`
  - `strms_prop_c`
- A queue named `streams_queue` is the source queue for all three propagations.
- There is a problem at the destination for the `strms_prop_a` propagation. This propagation cannot send messages to the destination queue.
- The other two propagations (`strms_prop_b` and `strms_prop_c`) are propagating messages normally.

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

Splitting and Merging an Oracle Streams Destination Automatically

Before reviewing this example, see the following sections:

- "Automatic Split and Merge" on page 12-12 for conceptual information
- "Examples That Split and Merge Oracle Streams" on page 12-14 for assumptions about the Oracle Streams replication environment in this example

Complete the following steps to split and merge a stream automatically:

1. In SQL*Plus, connect as the Oracle Streams administrator to the database with the capture process.

   See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.

2. Ensure that the following parameters are set properly for the `strms_capture` capture process to enable automatic split and merge:

   - `split_threshold`: Ensure that this parameter is not set to `INFINITE`. The default setting for this parameter is `1800` seconds.
   - `merge_threshold`: Ensure that this parameter is not set to a negative value. The default setting for this parameter is `60` seconds.

   To check the settings for these parameters, query the `DBA_CAPTURE_PARAMETERS` view. See *Oracle Streams Concepts and Administration* for instructions.

3. If you must reset one or both of the capture process parameters described in Step 2, then use Oracle Enterprise Manager or the `SET_PARAMETER` procedure in the `DBMS_CAPTURE_ADM` package to reset the parameters. See *Oracle Database 2 Day + Data Replication and Integration Guide* for instructions about using Oracle Enterprise Manager. See *Oracle Streams Concepts and Administration* for instructions about using the `SET_PARAMETER` procedure.

4. Monitor the `DBA_STREAMS_SPLIT_MERGE` view periodically to check whether an automatic split and merge operation is in process.

   When an automatic split occurs, certain components, such as the capture process, queue, and propagation, are cloned, and each is given a system-generated name. The `DBA_STREAMS_SPLIT_MERGE` view contains the name of each cloned component, and other information about the split and merge operation.

   Query the `DBA_STREAMS_SPLIT_MERGE` view to determine whether a stream has been split off from the original capture process:

```
COLUMN ORIGINAL_CAPTURE_NAME HEADING 'Original|Capture|Process' FORMAT A10
COLUMN ACTION_TYPE HEADING 'Action|Type' FORMAT A7
COLUMN STATUS_UPDATE_TIME HEADING 'Status|Update|Time' FORMAT A15
COLUMN STATUS HEADING 'Status' FORMAT A16
COLUMN JOB_NEXT_RUN_DATE HEADING 'Next Job|Run Date' FORMAT A20

SELECT ORIGINAL_CAPTURE_NAME,
       ACTION_TYPE,
       STATUS_UPDATE_TIME,
       STATUS,
       JOB_NEXT_RUN_DATE
  FROM DBA_STREAMS_SPLIT_MERGE
  ORDER BY STATUS_UPDATE_TIME DESC;
```

If a stream has been split off from the original capture process, then your output looks similar to the following:

```
Original          Status
Capture    Action  Update                          Next Job
Process    Type    Time            Status          Run Date
---------- ------- --------------- ---------------- --------------------
DB$CAP     MERGE   01-APR-09 06.49 NOTHING TO MERGE 01-APR-09 06.54.29.0
                   .29.204804 AM                    00000 AM -07:00
DB$CAP     SPLIT   01-APR-09 06.49 SPLIT DONE       01-APR-09 06.47.59.0
                   .17.389146 AM                    00000 AM -07:00
```

This output shows that an automatic split was performed. The merge job was run at `01-APR-09 06.49.29.204804 AM`, but the status shows `NOTHING TO MERGE` because the split stream is not ready to merge yet. The `SPLIT DONE` status indicates that the stream was split off at the following date and time: `01-APR-09 06.49.17.389146 AM`.

5. After an automatic split is performed, correct the problem with the destination. The problem is corrected when the apply process at the destination database can accept changes from the cloned capture process. An Oracle Scheduler job performs an automatic merge when the problem is corrected.
6. If the cloned capture process is disabled, then start the cloned capture process. The cloned capture process is disabled only if the stream is not a combined capture and apply optimization. See *Oracle Streams Concepts and Administration* for instructions for starting a capture process.

The cloned capture process captures changes that satisfy its rule sets. These changes are sent to the apply process.

During this time, an Oracle Scheduler job runs the `MERGE_STREAMS_JOB` procedure according to its schedule. The `MERGE_STREAMS_JOB` procedure queries the `CAPTURE_MESSAGE_CREATE_TIME` in the `GV$STREAMS_CAPTURE` view. When the difference between `CAPTURE_MESSAGE_CREATE_TIME` of the cloned capture process and the original capture process is less than or equal to the value of the `merge_threshold` capture process parameter, the `MERGE_STREAMS_JOB` procedure determines that the streams are ready to merge. The `MERGE_STREAMS_JOB` procedure runs the `MERGE_STREAMS` procedure automatically to merge the streams back together.

The `LAG` column in the `DBA_STREAMS_SPLIT_MERGE` view tracks the time in seconds that the cloned capture process lags behind the original capture process. The following query displays the lag time:

```
COLUMN ORIGINAL_CAPTURE_NAME HEADING 'Original Capture Process' FORMAT A25
COLUMN CLONED_CAPTURE_NAME HEADING 'Cloned Capture Process' FORMAT A25
COLUMN LAG HEADING 'Lag' FORMAT 999999999999999

SELECT ORIGINAL_CAPTURE_NAME,
       CLONED_CAPTURE_NAME,
       LAG
 FROM DBA_STREAMS_SPLIT_MERGE
 WHERE ACTION_TYPE = 'MERGE';
```

Your output looks similar to the following:

```
Original Capture Process  Cloned Capture Process                Lag
------------------------- ------------------------- ----------------
DB$CAP                    CLONED$_DB$CAP_5                      2048
```

This output shows that there is a lag of 2,048 seconds between the CAPTURE_MESSAGE_CREATE_TIME values for the original capture process and the cloned capture process. When the cloned capture process is within the threshold, the merge job can start the MERGE_STREAMS procedure. By default, the merge threshold is 60 seconds.

The MERGE_STREAMS procedure performs the following actions:

- Stops the cloned capture process.
- Re-creates the original propagation called strms_prop_a.
- Drops the cloned propagation.
- Drops the cloned capture process.
- Drops the cloned queue.

Repeat the query in Step 4 periodically to monitor the split and merge operation. After the merge operation is complete, the output for this query is similar to the following:

```
Original          Status
Capture    Action Update                          Next Job
Process    Type   Time            Status          Run Date
---------- ------- --------------- ---------------- --------------------
DB$CAP     MERGE  01-APR-09 07.32 NOTHING TO MERGE 01-APR-09 07.37.04.0
                  .04.820795 AM                    00000 AM -07:00
DB$CAP     MONITOR 01-APR-09 07.32 MERGE DONE      01-APR-09 07.36.20.0
                  .04.434925 AM                    00000 AM -07:00
DB$CAP     SPLIT  01-APR-09 06.49 SPLIT DONE       01-APR-09 06.47.59.0
                  .17.389146 AM                    00000 AM -07:00
```

This output shows that the split stream was merged back into the original capture process at the following date an time: 01-APR-09 07.32.04.434925 AM. The next status shows NOTHING TO MERGE because there are no remaining split streams.

After the streams are merged, the Oracle Streams replication environment has the same components as it had before the split and merge operation. Information about the completed split and merge operation is stored in the DBA_STREAMS_SPLIT_MERGE_HIST for future reference.

> **See Also:** *Oracle Streams Concepts and Administration* for information about monitoring automatic split and merge operations

Splitting an Oracle Streams Destination Manually and Merging It Automatically

Before reviewing this example, see the following sections:

- "Manual Split and Automatic Merge" on page 12-13 for conceptual information
- "Examples That Split and Merge Oracle Streams" on page 12-14 for assumptions about the Oracle Streams replication environment in this example

The example in this section splits the stream manually and merges it automatically. That is, the perform_actions parameter is set to TRUE in the SPLIT_STREAMS procedure. Also, the example merges the streams automatically at the appropriate time because the auto_merge_threshold parameter is to set a positive number (60) in the SPLIT_STREAMS procedure.

Complete the following steps to split streams directly and merge streams automatically:

1. In SQL*Plus, connect as the Oracle Streams administrator to the database with the capture process.

   See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.

2. Run the following procedure to split the stream flowing through propagation `strms_prop_a` from the other propagations flowing from the `strms_capture` capture process:

```
DECLARE
    schedule_name  VARCHAR2(30);
    job_name       VARCHAR2(30);
BEGIN
    schedule_name := 'merge_job1_schedule';
    job_name      := 'merge_job1';
  DBMS_STREAMS_ADM.SPLIT_STREAMS(
    propagation_name        => 'strms_prop_a',
    cloned_propagation_name => 'cloned_prop_a',
    cloned_queue_name       => 'cloned_queue',
    cloned_capture_name     => 'cloned_capture',
    perform_actions         => TRUE,
    auto_merge_threshold    => 60,
    schedule_name           => schedule_name,
    merge_job_name          => job_name);
END;
/
```

   Running this procedure performs the following actions:

   - Creates a new queue called `cloned_queue`.
   - Creates a new propagation called `cloned_prop_a` that propagates messages from the `cloned_queue` queue to the existing destination queue used by the `strms_prop_a` propagation. The cloned propagation `cloned_prop_a` uses the same rule set as the original propagation `strms_prop_a`.
   - Stops the capture process `strms_capture`.
   - Queries the acknowledge SCN for the original propagation `strms_prop_a`. The acknowledged SCN is the last SCN acknowledged by the apply process that applies the changes sent by the propagation. The `ACKED_SCN` value in the `DBA_PROPAGATION` view shows the acknowledged SCN for a propagation.
   - Creates a new capture process called `cloned_capture`. The start SCN for `cloned_capture` is set to the value of the acknowledged SCN for the `strms_prop_a` propagation. The cloned capture process `cloned_capture` uses the same rule set as the original capture process `strms_capture`.
   - Drops the original propagation `strms_prop_a`.
   - Starts the original capture process `strms_capture` with the start SCN set to the value of the acknowledged SCN for the `strms_prop_a` propagation.
   - Creates an Oracle Scheduler job named `merge_job1` with a schedule named `merge_job1_schedule`. Both the job and the schedule are owned by the user who ran the `SPLIT_STREAMS` procedure. The schedule starts to run when the `SPLIT_STREAMS` procedure completes. The system defines the initial schedule, but you can modify it in the same way that you would modify any Oracle Scheduler job. See *Oracle Database Administrator's Guide* for instructions.

3. Correct the problem with the destination of `cloned_prop_a`. The problem is corrected when the apply process at the destination database can accept changes from the cloned capture process.
4. While connected as the Oracle Streams administrator, start the cloned capture process by running the following procedure:

```
exec DBMS_CAPTURE_ADM.START_CAPTURE('cloned_capture');
```

After the cloned capture process `cloned_capture` starts running, it captures changes that satisfy its rule sets from the acknowledged SCN forward. These changes are propagated by the `cloned_prop_a` propagation and processed by the apply process at the destination database.

During this time, the Oracle Scheduler job runs the `MERGE_STREAMS_JOB` procedure according to its schedule. The `MERGE_STREAMS_JOB` procedure queries the `CAPTURE_MESSAGE_CREATE_TIME` in the `GV$STREAMS_CAPTURE` view. When the difference between `CAPTURE_MESSAGE_CREATE_TIME` of the cloned capture process `cloned_capture` and the original capture process `strms_capture` is less than or equal 60 seconds, the `MERGE_STREAMS_JOB` procedure determines that the streams are ready to merge. The `MERGE_STREAMS_JOB` procedure runs the `MERGE_STREAMS` procedure automatically to merge the streams back together.

The following query displays the `CAPTURE_MESSAGE_CREATE_TIME` for the original capture process and cloned capture process:

```
COLUMN CAPTURE_NAME HEADING 'Capture|Name' FORMAT A17
COLUMN STATE HEADING 'State' FORMAT A20
COLUMN CREATE_MESSAGE HEADING 'Last Message|Create Time'

SELECT CAPTURE_NAME,
 STATE,
 TO_CHAR(CAPTURE_MESSAGE_CREATE_TIME, 'HH24:MI:SS MM/DD/YY') CREATE_MESSAGE
 FROM V$STREAMS_CAPTURE;
```

Your output looks similar to the following:

```
Capture                                Last Message
Name              State                Create Time
----------------- -------------------- -----------------
DB$CAP            CAPTURING CHANGES    07:22:55 04/01/09
CLONED$_DB$CAP_5  CAPTURING CHANGES    06:50:39 04/01/09
```

This output shows that there is more than a 30 minute difference between the `CAPTURE_MESSAGE_CREATE_TIME` values for the original capture process and the cloned capture process. When the cloned capture process is within the threshold, the merge job can start the `MERGE_STREAMS` procedure. By default, the merge threshold is 60 seconds.

The `MERGE_STREAMS` procedure performs the following actions:

- Stops the cloned capture process `cloned_capture`.
- Re-creates the propagation called `strms_prop_a`.
- Drops the cloned propagation `cloned_prop_a`.
- Drops the cloned capture process `cloned_capture`.
- Drops the cloned queue `cloned_queue`.

After the streams are merged, the Oracle Streams replication environment has the same components as it had before the split and merge operation. Information about

the completed split and merge operation is stored in the DBA_STREAMS_SPLIT_MERGE_HIST for future reference.

Splitting and Merging an Oracle Streams Destination Manually With Scripts

Before reviewing this example, see the following sections:

- "Manual Split and Merge With Generated Scripts" on page 12-13 for conceptual information
- "Examples That Split and Merge Oracle Streams" on page 12-14 for assumptions about the Oracle Streams replication environment in this example

The example in this section splits and merges streams by generating and running scripts. That is, the perform_actions parameter is set to FALSE in the SPLIT_STREAMS procedure. Also, the example merges the streams manually at the appropriate time because the auto_merge_threshold parameter is set to NULL in the SPLIT_STREAMS procedure.

Complete the following steps to use scripts to split and merge streams:

1. In SQL*Plus, connect as the Oracle Streams administrator to the database with the capture process.

   See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.

2. If it does not already exist, then create a directory object named db_dir to hold the scripts generated by the procedures:

   ```
   CREATE DIRECTORY db_dir AS '/usr/db_files';
   ```

3. Run the following procedure to generate a script to split the streams:

   ```
   DECLARE
       schedule_name  VARCHAR2(30);
       job_name       VARCHAR2(30);
   BEGIN
     DBMS_STREAMS_ADM.SPLIT_STREAMS(
       propagation_name        => 'strms_prop_a',
       cloned_propagation_name => 'cloned_prop_a',
       cloned_queue_name       => 'cloned_queue',
       cloned_capture_name     => 'cloned_capture',
       perform_actions         => FALSE,
       script_name             => 'split.sql',
       script_directory_object => 'db_dir',
       auto_merge_threshold    => NULL,
       schedule_name           => schedule_name,
       merge_job_name          => job_name);
   END;
   /
   ```

   Running this procedure generates the split.sql script. The script contains the actions that will split the stream flowing through propagation strms_prop_a from the other propagations flowing from the strms_capture capture process.

4. Go to the directory used by the db_dir directory object, and open the split.sql script with a text editor.
5. Examine the script and make modifications, if necessary.
6. Save and close the script.
7. While connected as the Oracle Streams administrator in SQL*Plus, run the script:

```
@/usr/db_files/split.sql
```

Running the script performs the following actions:

- Runs the SET_UP_QUEUE procedure in the DBMS_STREAMS_ADM package to create a queue called cloned_queue.
- Runs the CREATE_PROPAGATION procedure in the DBMS_PROPAGATION_ADM package to create a propagation called cloned_prop_a. This new propagation propagates messages from the cloned_queue queue to the existing destination queue used by the strms_prop_a propagation. The cloned propagation cloned_prop_a uses the same rule set as the original propagation strms_prop_a.

  The CREATE_PROPAGATION procedure sets the original_propagation_name parameter to strms_prop_a and the auto_merge_threshold parameter to NULL.
- Runs the STOP_CAPTURE procedure in the DBMS_CAPTURE_ADM package to stop the capture process strms_capture.
- Queries the acknowledge SCN for the original propagation strms_prop_a. The acknowledged SCN is the last SCN acknowledged by the apply process that applies the changes sent by the propagation. The ACKED_SCN value in the DBA_PROPAGATION view shows the acknowledged SCN for a propagation.
- Runs the CREATE_CAPTURE procedure in the DBMS_CAPTURE_ADM package to create a capture process called cloned_capture. The start SCN for cloned_capture is set to the value of the acknowledged SCN for the strms_prop_a propagation. The cloned capture process cloned_capture uses the same rule set as the original capture process strms_capture.
- Runs the DROP_PROPAGATION procedure in the DBMS_PROPAGATION_ADM package to drop the original propagation strms_prop_a.
- Runs the START_CAPTURE procedure in the DBMS_CAPTURE_ADM package to start the original capture process strms_capture with the start SCN set to the value of the acknowledged SCN for the strms_prop_a propagation.

8. Correct the problem with the destination of cloned_prop_a. The problem is corrected when the apply process at the destination database can accept changes from the cloned capture process.
9. While connected as the Oracle Streams administrator, start the cloned capture process by running the following procedure:

   ```
   exec DBMS_CAPTURE_ADM.START_CAPTURE('cloned_capture');
   ```

10. Monitor the Oracle Streams replication environment until the cloned capture process catches up to, or nearly catches up to, the original capture process. Specifically, query the CAPTURE_MESSAGE_CREATION_TIME column in the GV$STREAMS_CAPTURE view for each capture process.

    Run the following query to check the CAPTURE_MESSAGE_CREATE_TIME for each capture process periodically:

    ```
    SELECT CAPTURE_NAME,
           TO_CHAR(CAPTURE_MESSAGE_CREATE_TIME, 'HH24:MI:SS MM/DD/YY')
       FROM GV$STREAMS_CAPTURE;
    ```

Do not move on to the next step until the difference between CAPTURE_MESSAGE_CREATE_TIME of the cloned capture process cloned_capture and the original capture process strms_capture is relatively small.

11. Run the following procedure to generate a script to merge the streams:

```
BEGIN
  DBMS_STREAMS_ADM.MERGE_STREAMS(
    cloned_propagation_name => 'cloned_prop_a',
    perform_actions         => FALSE,
    script_name             => 'merge.sql',
    script_directory_object => 'db_dir');
END;
/
```

Running this procedure generates the merge.sql script. The script contains the actions that will merge the stream flowing through propagation cloned_prop_a with the other propagations flowing from the strms_capture capture process.

12. Go to the directory used by the db_dir directory object, and open the merge.sql script with a text editor.
13. Examine the script and make modifications, if necessary.
14. Save and close the script.
15. While connected as the Oracle Streams administrator in SQL*Plus, run the script:

```
@/usr/db_files/merge.sql
```

Running the script performs the following actions:

- Runs the STOP_CAPTURE procedure in the DBMS_CAPTURE_ADM package to stop the cloned capture process cloned_capture.
- Runs the STOP_CAPTURE procedure in the DBMS_CAPTURE_ADM package to stop the original capture process strms_capture.
- Runs the CREATE_PROPAGATION procedure in the DBMS_PROPAGATION_ADM package to re-create the propagation called strms_prop_a.
- Starts the original capture process strms_capture from the lower SCN value of these two SCN values:
  - The acknowledged SCN of the cloned propagation cloned_prop_a.
  - The lowest acknowledged SCN of the other propagations that propagate changes captured by the original capture process (propagations strms_prop_b and strms_prop_c in this example).

  When the strms_capture capture process is started, it might recapture changes that it already captured, or it might capture changes that were already captured by the cloned capture process cloned_capture. In either case, the relevant apply processes will discard any duplicate changes they receive.
- Runs the DROP_PROPAGATION procedure in the DBMS_PROPAGATION_ADM package to drop the cloned propagation cloned_prop_a.
- Runs the DROP_CAPTURE procedure in the DBMS_CAPTURE_ADM package to drop the cloned capture process cloned_capture.
- Runs the REMOVE_QUEUE procedure in the DBMS_STREAMS_ADM package to drop the cloned queue cloned_queue.

After the script runs successfully, the streams are merged, and the Oracle Streams replication environment has the same components as it had before the split and merge operation. Information about the completed split and merge operation is stored in the DBA_STREAMS_SPLIT_MERGE_HIST for future reference.

Changing the DBID or Global Name of a Source Database

Typically, database administrators change the DBID and global name of a database when it is a clone of another database. You can view the DBID of a database by querying the DBID column in the V$DATABASE dynamic performance view, and you can view the global name of a database by querying the GLOBAL_NAME static data dictionary view. When you change the DBID or global name of a source database, any existing capture processes that capture changes originating at this source database become unusable. The capture processes can be local capture processes or downstream capture processes that capture changes that originated at the source database. Also, any existing apply processes that apply changes from the source database become unusable. However, existing synchronous captures and propagations do not need to be re-created, although modifications to propagation rules might be necessary.

If a capture process or synchronous capture is capturing changes to a source database for which you have changed the DBID or global name, then complete the following steps:

1. Shut down the source database.
2. Restart the source database with RESTRICTED SESSION enabled using STARTUP RESTRICT.
3. Drop the capture process using the DROP_CAPTURE procedure in the DBMS_CAPTURE_ADM package. The capture process can be a local capture process at the source database or a downstream capture process at a remote database. Synchronous captures do not need to be dropped.
4. At the source database, run the ALTER SYSTEM SWITCH LOGFILE statement on the database.
5. If any changes have been captured from the source database, 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. Modify any rules that use the source database name as a condition. The source database name should be changed to the new global name of the source database where appropriate in these rules. You might need to modify capture process rules, propagation rules, and apply process rules at the local database and at remote databases in the environment. Typically, synchronous capture rules do not contain a condition for the source database.
7. Drop the apply processes that apply changes from the capture process that you dropped in Step 3. Use the DROP_APPLY procedure in the DBMS_APPLY_ADM package to drop an apply process. Apply processes that apply changes captured by synchronous capture do not need to be dropped.
8. At each destination database that applies changes from the source database, re-create the apply processes you dropped in Step 7. You might want to associate the each apply process with the same rule sets it used before it was dropped. See Chapter 7, "Configuring Implicit Apply" for instructions.
9. Re-create the capture process you dropped in Step 3, if necessary. You might want to associate the capture process with the same rule sets used by the capture

process you dropped in Step 3. See "Configuring a Capture Process" on page 5-1 for instructions.

10. At the source database, prepare database objects whose changes will be captured by the re-created capture process for instantiation. See "Preparing Database Objects for Instantiation at a Source Database" on page 8-8.
11. At each destination database that applies changes from the source database, set the instantiation SCN for all databases objects to which changes from the source database will be applied. See "Setting Instantiation SCNs at a Destination Database" on page 8-36 for instructions.
12. Disable the restricted session using the `ALTER SYSTEM DISABLE RESTRICTED SESSION` statement.
13. At each destination database that applies changes from the source database, start the apply processes you created in Step 8.
14. At the source database, start the capture process you created in Step 9.

> **See Also:** *Oracle Database Utilities* for more information about changing the `DBID` of a database using the DBNEWID utility

Resynchronizing a Source Database in a Multiple-Source Environment

A multiple-source environment is one in which there is more than one source database for any of the shared data. If a source database in a multiple-source environment cannot be recovered to the current point in time, then you can use the method described in this section to resynchronize the source database with the other source databases in the environment. Some reasons why a database cannot be recovered to the current point in time include corrupted archived redo logs or the media failure of an online redo log group.

For example, a bidirectional Oracle Streams environment is one in which exactly two databases share the replicated database objects and data. In this example, assume that database A is the database that must be resynchronized and that database B is the other source database in the environment. To resynchronize database A in this bidirectional Oracle Streams environment, complete the following steps:

1. Verify that database B has applied all of the changes sent from database A. You can query the `V$BUFFERED_SUBSCRIBERS` data dictionary view at database B to determine whether the apply process that applies these changes has any unapplied changes in its queue. See the example on viewing propagations dequeuing LCRs from each buffered queue in *Oracle Streams Concepts and Administration* for an example of such a query. Do not continue until all of these changes have been applied.
2. Remove the Oracle Streams configuration from database A by running the `REMOVE_STREAMS_CONFIGURATION` procedure in the `DBMS_STREAMS_ADM` package. See *Oracle Database PL/SQL Packages and Types Reference* for more information about this procedure.
3. At database B, drop the apply process that applies changes from database A. Do not drop the rule sets used by this apply process because you will re-create the apply process in a subsequent step.
4. Complete the steps in "Adding a New Database to an Existing Multiple-Source Environment" on page 4-25 to add database A back into the Oracle Streams environment.

Performing Database Point-in-Time Recovery in an Oracle Streams Environment

Point-in-time recovery is the recovery of a database to a specified noncurrent time, SCN, or log sequence number. The following sections discuss performing point-in-time recovery in an Oracle Streams replication environment:

- Performing Point-in-Time Recovery on the Source in a Single-Source Environment
- Performing Point-in-Time Recovery in a Multiple-Source Environment
- Performing Point-in-Time Recovery on a Destination Database

> **See Also:** *Oracle Database Backup and Recovery User's Guide* for more information about point-in-time recovery

Performing Point-in-Time Recovery on the Source in a Single-Source Environment

A single-source Oracle Streams replication environment is one in which there is only one source database for shared data. If database point-in-time recovery is required at the source database in a single-source Oracle Streams environment, and any capture processes that capture changes generated at a source database are running, then you must stop these capture processes before you perform the recovery operation. Both local and downstream capture process that capture changes generated at the source database must be stopped. 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.

The instructions in this section assume that the single-source replication environment has the following characteristics:

- Only one capture process named `strm01_capture`, which can be a local or downstream capture process
- Only one destination database with the global name `dest.example.com`
- Only one apply process named `strm01_apply` at the destination database

If point-in-time recovery must be performed on the source database, then you can follow these instructions to recover as many transactions as possible at the source database by using transactions applied at the destination database. These instructions assume that you can identify the transactions applied at the destination database after the source point-in-time SCN and execute these transactions at the source database.

Note: Oracle recommends that you set the apply process parameter `commit_serialization` to `FULL` when performing point-in-time recovery in a single-source Oracle Streams replication environment.

Complete the following steps to perform point-in-time recovery on the source database in a single-source Oracle Streams replication environment:

1. Perform point-in-time recovery on the source database if you have not already done so. Note the point-in-time recovery SCN because it is needed in subsequent steps.
2. Ensure that the source database is in restricted mode.

3. Connect to the database running the capture process and list the rule sets used by the capture process.

   To list the rule sets used by the capture process, run the following query:

```
COLUMN CAPTURE_NAME HEADING 'Capture|Process|Name' FORMAT A15
COLUMN RULE_SET_OWNER HEADING 'Positive|Rule Owner' FORMAT A15
COLUMN RULE_SET_NAME HEADING 'Positive|Rule Set' FORMAT A15
COLUMN NEGATIVE_RULE_SET_OWNER HEADING 'Negative|Rule Owner' FORMAT A15
COLUMN NEGATIVE_RULE_SET_NAME HEADING 'Negative|Rule Set' FORMAT A15

SELECT CAPTURE_NAME,
       RULE_SET_OWNER,
       RULE_SET_NAME,
       NEGATIVE_RULE_SET_OWNER,
       NEGATIVE_RULE_SET_NAME
   FROM DBA_CAPTURE;
```

   Make a note of the rule sets used by the capture process. You will need to specify these rule sets for the new capture process in Step 12.

4. Connect to the destination database and list the rule sets used by the apply process.

   To list the rule sets used by the capture process, run the following query:

```
COLUMN APPLY_NAME HEADING 'Apply|Process|Name' FORMAT A15
COLUMN RULE_SET_OWNER HEADING 'Positive|Rule Owner' FORMAT A15
COLUMN RULE_SET_NAME HEADING 'Positive|Rule Set' FORMAT A15
COLUMN NEGATIVE_RULE_SET_OWNER HEADING 'Negative|Rule Owner' FORMAT A15
COLUMN NEGATIVE_RULE_SET_NAME HEADING 'Negative|Rule Set' FORMAT A15

SELECT APPLY_NAME,
       RULE_SET_OWNER,
       RULE_SET_NAME,
       NEGATIVE_RULE_SET_OWNER,
       NEGATIVE_RULE_SET_NAME
   FROM DBA_APPLY;
```

   Make a note of the rule sets used by the apply process. You will need to specify these rule sets for the new apply process in Step k.

5. Stop the capture process using the `STOP_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package.

6. At the source database, perform a data dictionary build:

```
SET SERVEROUTPUT ON
DECLARE
  scn  NUMBER;
BEGIN
  DBMS_CAPTURE_ADM.BUILD(
    first_scn => scn);
  DBMS_OUTPUT.PUT_LINE('First SCN Value = ' || scn);
END;
/
```

   Note the SCN value returned because it is needed in Step 12.

7. At the destination database, wait until all of the transactions from the source database in the apply process's queue have been applied. The apply processes should become idle when these transactions have been applied. You can query the `STATE` column in both the `V$STREAMS_APPLY_READER` and `V$STREAMS_`

APPLY_SERVER. The state should be IDLE for the apply process in both views before you continue.

8. Perform a query at the destination database to determine the highest SCN for a transaction that was applied.

   If the apply process is running, then perform the following query:

```
SELECT HWM_MESSAGE_NUMBER FROM V$STREAMS_APPLY_COORDINATOR
  WHERE APPLY_NAME = 'STRM01_APPLY';
```

   If the apply process is disabled, then perform the following query:

```
SELECT APPLIED_MESSAGE_NUMBER FROM DBA_APPLY_PROGRESS
  WHERE APPLY_NAME = 'STRM01_APPLY';
```

   Note the highest apply SCN returned by the query because it is needed in subsequent steps.

9. If the highest apply SCN obtained in Step 8 is less than the point-in-time recovery SCN noted in Step 1, then proceed to Step 10. Otherwise, if the highest apply SCN obtained in Step 8 is greater than or equal to the point-in-time recovery SCN noted in Step 1, then the apply process has applied some transactions from the source database after point-in-time recovery SCN, and you must complete the following steps:

   a. Manually execute the transactions that were applied after the point-in-time SCN at the source database. When you execute these transactions at the source database, ensure that you set an Oracle Streams tag in the session so that the transactions will not be captured by the capture process. If no such Oracle Streams session tag is set, then these changes can be cycled back to the destination database. See "Managing Oracle Streams Tags for the Current Session" on page 10-16 for instructions.

   b. Disable the restricted session at the source database.

   c. Proceed to Step 11. Do not complete Step 10.

10. If the highest apply SCN obtained in Step 8 is less than the point-in-time recovery SCN noted in Step 1, then the apply process has not applied any transactions from the source database after point-in-time recovery SCN, and you must complete the following steps:

    a. Disable the restricted session at the source database.

    b. Ensure that the apply process is running at the destination database.

    c. Set the maximum_scn capture process parameter of the original capture process to the point-in-time recovery SCN using the SET_PARAMETER procedure in the DBMS_CAPTURE_ADM package.

    d. Set the start SCN of the original capture process to the oldest SCN of the apply process. You can determine the oldest SCN of a running apply process by querying the OLDEST_SCN_NUM column in the V$STREAMS_APPLY_READER dynamic performance view at the destination database. To set the start SCN of the capture process, specify the start_scn parameter when you run the ALTER_CAPTURE procedure in the DBMS_CAPTURE_ADM package.

    e. Ensure that the capture process writes information to the alert log by running the following procedure:

```
BEGIN
  DBMS_CAPTURE_ADM.SET_PARAMETER(
    capture_name => 'strm01_capture',
    parameter    => 'write_alert_log',
    value        => 'Y');
END;
/
```

f. Start the original capture process using the `START_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package.

g. Ensure that the original capture process has captured all changes up to the `maximum_scn` setting by querying the `CAPTURED_SCN` column in the `DBA_CAPTURE` data dictionary view. When the value returned by the query is equal to or greater than the `maximum_scn` value, the capture process should stop automatically. When the capture process is stopped, proceed to the next step.

h. Find the value of the `LAST_ENQUEUE_MESSAGE_NUMBER` in the alert log. Note this value because it is needed in subsequent steps.

i. At the destination database, wait until all the changes are applied. You can monitor the applied changes for the apply process `strm01_apply` by running the following queries at the destination database:

```
SELECT DEQUEUED_MESSAGE_NUMBER
  FROM V$STREAMS_APPLY_READER
  WHERE APPLY_NAME = 'STRM01_APPLY' AND
        DEQUEUED_MESSAGE_NUMBER = last_enqueue_message_number;
```

Substitute the `LAST_ENQUEUE_MESSAGE_NUMBER` found in the alert log in Step h for *last_enqueue_message_number* on the last line of the query. When this query returns a row, all of the changes from the capture database have been applied at the destination database.

Also, ensure that the state of the apply process reader server and each apply server is `IDLE`. For example, run the following queries for an apply process named `strm01_apply`:

```
SELECT STATE FROM V$STREAMS_APPLY_READER
  WHERE APPLY_NAME = 'STRM01_APPLY';

SELECT STATE FROM V$STREAMS_APPLY_SERVER
  WHERE APPLY_NAME = 'STRM01_APPLY';
```

When both of these queries return `IDLE`, move on to the next step.

j. At the destination database, drop the apply process using the `DROP_APPLY` procedure in the `DBMS_APPLY_ADM` package.

k. At the destination database, create a new apply process. The new apply process should use the same queue and rule sets used by the original apply process.

l. At the destination database, start the new apply process using the `START_APPLY` procedure in the `DBMS_APPLY_ADM` package.

11. Drop the original capture process using the `DROP_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package.

12. Create a new capture process using the `CREATE_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package to replace the capture process you dropped in Step 11. Specify the SCN returned by the data dictionary build in Step 6 for both

the `first_scn` and `start_scn` parameters. The new capture process should use the same queue and rule sets as the original capture process.

13. Start the new capture process using the `START_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package.

Performing Point-in-Time Recovery in a Multiple-Source Environment

A multiple-source environment is one in which there is more than one source database for any of the shared data. If database point-in-time recovery is required at a source database in a multiple-source Oracle Streams environment, then you can use another source database in the environment to recapture the changes made to the recovered source database after the point-in-time recovery.

For example, in a multiple-source Oracle Streams environment, one source database can become unavailable at time T2 and undergo point in time recovery to an earlier time T1. After recovery to T1, transactions performed at the recovered database between T1 and T2 are lost at the recovered database. However, before the recovered database became unavailable, assume that these transactions were propagated to another source database and applied. In this case, you can use this other source database to restore the lost changes to the recovered database.

Specifically, to restore changes made to the recovered database after the point-in-time recovery, you configure a capture process to recapture these changes from the redo logs at the other source database, a propagation to propagate these changes from the database where changes are recaptured to the recovered database, and an apply process at the recovered database to apply these changes.

Changes originating at the other source database that were applied at the recovered database between T1 and T2 also have been lost and must be recovered. To accomplish this, alter the capture process at the other source database to start capturing changes at an earlier SCN. This SCN is the oldest SCN for the apply process at the recovered database.

The following SCN values are required to restore lost changes to the recovered database:

- **Point-in-time SCN**: The SCN for the point-in-time recovery at the recovered database.
- **Instantiation SCN**: The SCN value to which the instantiation SCN must be set for each database object involved in the recovery at the recovered database while changes are being reapplied. At the other source database, this SCN value corresponds to one less than the *commit SCN* of the first transaction that was applied at the other source database and lost at the recovered database.
- **Start SCN**: The SCN value to which the start SCN is set for the capture process created to recapture changes at the other source database. This SCN value corresponds to the *earliest SCN* at which the apply process at the other source database started applying a transaction that was lost at the recovered database. This capture process can be a local or downstream capture process that uses the other source database for its source database.
- **Maximum SCN**: The SCN value to which the `maximum_scn` parameter for the capture process created to recapture lost changes should be set. The capture process stops capturing changes when it reaches this SCN value. The current SCN for the other source database is used for this value.

You should record the point-in-time SCN when you perform point-in-time recovery on the recovered database. You can use the `GET_SCN_MAPPING` procedure in the `DBMS_STREAMS_ADM` package to determine the other necessary SCN values.

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

Performing Point-in-Time Recovery on a Destination Database

If database point-in-time recovery is required at a destination database in an Oracle Streams environment, then you must reapply the captured changes that had already been applied after the point-in-time 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 an Oracle 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 might 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 persistent LCRs to the destination database. Both of these methods reapply only captured LCRs at the destination database, not persistent LCRs.
- In a directed networks configuration, the destination database is used to propagate LCRs from a capture process to other databases, but the destination database does not apply LCRs from this capture process.
- The oldest message number for an apply process at the destination database is lower than the first SCN of a capture process that captures changes for this apply process. The following query at a destination database lists the oldest message number (oldest SCN) for each apply process:

  ```
  SELECT APPLY_NAME, OLDEST_MESSAGE_NUMBER FROM DBA_APPLY_PROGRESS;
  ```

The following query at a source database lists the first SCN for each capture process:

```
SELECT CAPTURE_NAME, FIRST_SCN FROM DBA_CAPTURE;
```

- The archived log files that contain the intended start SCN are no longer available.

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.

Note: If you are using combined capture and apply in a single-source replication environment, and the destination database has undergone point-in-time recovery, then the Oracle Streams capture process automatically detects where to capture changes upon restart, and no extra steps are required for it. See *Oracle Streams Concepts and Administration* for more information.

See Also: *Oracle Streams Concepts and Administration* for more information about SCN values relating to a capture process and directed networks

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 the destination database is also a source database in a multiple-source Oracle Streams environment, then complete the actions described in "Performing Point-in-Time Recovery in a Multiple-Source Environment" on page 12-29.

2. 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.

   Do not drop the rule sets used by the propagations you drop.

   If the existing capture process is a downstream capture process that is configured at the destination database, then the downstream capture process is recovered to the same point-in-time as the destination database when you perform point-in-time recovery in Step 3. In this case, the remaining steps in this section after Step 3 are not required. Ensure that the required redo log files are available to the capture process.

   Note: You must drop the appropriate propagation(s). Disabling them is not sufficient. You will re-create the propagation(s) in Step 7, and dropping them now ensures that only LCRs created after resetting the start SCN for the capture process are propagated.

> **See Also:** *Oracle Streams Concepts and Administration* for more information about directed networks

3. Perform the point-in-time recovery at the destination database.
4. Query for the oldest message number (oldest SCN) from the source database for the apply process at the destination database. Make a note of the results of the query. The oldest message number is the earliest system change number (SCN) that might need to be applied.

   The following query at a destination database lists the oldest message number for each apply process:

   ```
   SELECT APPLY_NAME, OLDEST_MESSAGE_NUMBER FROM DBA_APPLY_PROGRESS;
   ```

5. Stop the existing capture process using the `STOP_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package.
6. 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 4. 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;
   /
   ```

7. 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 any rule sets used by the original propagation 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.
8. 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 capture process to perform point-in-time recovery, then complete the following steps:

1. If the destination database is also a source database in a multiple-source Oracle Streams environment, then complete the actions described in "Performing Point-in-Time Recovery in a Multiple-Source Environment" on page 12-29.
2. 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 LCRs 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: *Oracle Streams Concepts and Administration* for more information about directed networks

3. Perform the point-in-time recovery at the destination database.
4. Query for the oldest message number (oldest SCN) from the source database for the apply process at the destination database. Make a note of the results of the query. The oldest message number is the earliest system change number (SCN) that might need to be applied.

   The following query at a destination database lists the oldest message number for each apply process:

   ```
   SELECT APPLY_NAME, OLDEST_MESSAGE_NUMBER FROM DBA_APPLY_PROGRESS;
   ```

5. 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.

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 created in Step 5 to the destination queue using the `CREATE_PROPAGATION` procedure in the `DBMS_PROPAGATION_ADM` package. Specify any rule sets used by the original propagation 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 7 between the queues created in Step 5.

7. 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 5 and the `start_scn` parameter to the value you recorded from the query in Step 4. Also, specify any rule sets used by the original capture process. If the rule sets used by the original capture process instruct the capture process to capture changes that should not be sent to the destination database that was recovered, then you can create and use smaller, customized rule sets that share some rules with the original rule sets.
8. Start the capture process you created in Step 7 using the `START_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package.

9. 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;
   ```

10. 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 7.
11. If you are not using directed networks between the source database and destination database, then drop the new propagation created in Step 6.

    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.
12. If you are not using directed networks between the source database and destination database, then remove the queue 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 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.
13. 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 any rule sets used by the original propagation 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 2.
14. Start the capture process you stopped in Step 9.

All of the steps after Step 8 can be deferred to a later time, or they can be done as soon as the condition described in Step 9 is met.

Running Flashback Queries in an Oracle Streams Replication Environment

Oracle Flashback Query enables you to view and repair historical data. You can perform queries on a database as of a certain clock time or system change number (SCN). In an Oracle Streams single-source replication environment, you can use Flashback Query at the source database and a destination database at a past time when the replicated database objects should be identical.

You can run the queries at corresponding SCNS at the source and destination databases to determine whether all of the changes to the replicated objects performed at the source database have been applied at the destination database. If there are apply errors at the destination database, then such a Flashback Query can show how the replicated objects looked at the time when the error was raised. This information could be useful in determining the cause of the error and the best way to correct the error.

Running a Flashback Query at each database can also check whether tables have certain rows at the corresponding SCNs. If the table data does not match at the corresponding SCNs, then there is a problem with the replication environment.

To run queries, the Oracle Streams replication environment must have the following characteristics:

- The replication environment must be a single-source environment, where changes to replicated objects are captured at only one database.
- No modifications are made to the replicated objects in the Stream. That is, no transformations, subset rules (row migration), or apply handlers modify the LCRs for the replicated objects.
- No DML or DDL changes are made to the replicated objects at the destination database.
- Both the source database and the destination database must be configured to use Oracle Flashback, and the Oracle Streams administrator at both databases must be able to execute subprograms in the `DBMS_FLASHBACK` package.
- The information in the undo tablespace must go back far enough to perform the query at each database. Oracle Flashback features use the Automatic Undo Management system to obtain historical data and metadata for a transaction. The `UNDO_RETENTION` initialization parameter at each database must be set to a value that is large enough to perform the Flashback Query.

Because Oracle Streams replication is asynchronous, you cannot use a past time in the Flashback Query. However, you can use the `GET_SCN_MAPPING` procedure in the `DBMS_STREAMS_ADM` package to determine the SCN at the destination database that corresponds to an SCN at the source database.

These instructions assume that you know the SCN for the Flashback Query at the source database. Using this SCN, you can determine the corresponding SCN for the Flashback Query at the destination database. To run these queries, complete the following steps:

1. At the destination database, ensure that the archived redo log file for the approximate time of the Flashback Query is available to the database. The `GET_SCN_MAPPING` procedure requires that this redo log file be available.
2. In SQL*Plus, connect to the destination database as the Oracle Streams administrator.

   See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.
3. Run the `GET_SCN_MAPPING` procedure. In this example, assume that the SCN for the source database is `52073983` and that the name of the apply process that applies changes from the source database is `strm01_apply`:

```
SET SERVEROUTPUT ON
DECLARE
  dest_scn   NUMBER;
  start_scn  NUMBER;
  dest_skip  DBMS_UTILITY.NAME_ARRAY;
```

```
BEGIN
  DBMS_STREAMS_ADM.GET_SCN_MAPPING(
    apply_name             => 'strm01_apply',
    src_pit_scn            => '52073983',
    dest_instantiation_scn => dest_scn,
    dest_start_scn         => start_scn,
    dest_skip_txn_ids      => dest_skip);
  IF dest_skip.count = 0 THEN
    DBMS_OUTPUT.PUT_LINE('No Skipped Transactions');
    DBMS_OUTPUT.PUT_LINE('Destination SCN: ' || dest_scn);
  ELSE
    DBMS_OUTPUT.PUT_LINE('Destination SCN invalid for Flashback Query.');
    DBMS_OUTPUT.PUT_LINE('At least one transaction was skipped.');
  END IF;
END;
/
```

If a valid destination SCN is returned, then proceed to Step 4.

If the destination SCN was not valid for Flashback Query because one or more transactions were skipped by the apply process, then the apply process parameter `commit_serialization` was set to `DEPENDENT_TRANSACTIONS`, and nondependent transactions have been applied out of order. There is at least one transaction with a source commit SCN less than `src_pit_scn` that was committed at the destination database after the returned `dest_instantiation_scn`. Therefore, tables might not be the same at the source and destination databases for the specified source SCN. You can choose a different source SCN and restart at Step 1.

4. Run the Flashback Query at the source database using the source SCN.
5. Run the Flashback Query at the destination database using the SCN returned in Step 3.
6. Compare the results of the queries in Steps 4 and 5 and take any necessary action.

See Also:

- *Oracle Database Advanced Application Developer's Guide* for more information about Flashback Query
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `GET_SCN_MAPPING` procedure

Recovering from Operation Errors

You can recover from the following operations using the `RECOVER_OPERATION` procedure in the `DBMS_STREAMS_ADM` package:

- Split and merge operations using:
  - Automatic split and merge operations invoked by the `split_threshold` and `merge_threshold` capture process parameters.
  - Spit and merge operations that use the `SPLIT_STREAMS` and `MERGE_STREAMS_JOB` procedures in the `DBMS_STREAMS_ADM` package.
- Change table configuration operations performed by the `MAINTAIN_CHANGE_TABLE` procedure in the `DBMS_STREAMS_ADM` package.
- Replication configuration operations performed by the following procedures in the `DBMS_STREAMS_ADM` package:

  - MAINTAIN_GLOBAL
  - MAINTAIN_SCHEMAS
  - MAINTAIN_SIMPLE_TTS
  - MAINTAIN_TABLES
  - MAINTAIN_TTS
  - PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP

Information about the operation is stored in the following data dictionary views when the operation is in process:

- DBA_RECOVERABLE_SCRIPT
- DBA_RECOVERABLE_SCRIPT_HIST
- DBA_RECOVERABLE_SCRIPT_PARAMS
- DBA_RECOVERABLE_SCRIPT_BLOCKS
- DBA_RECOVERABLE_SCRIPT_ERRORS

> **Note:** If the perform_actions parameter is set to FALSE when one of the configuration procedures is run, and a script is used to configure the Oracle Streams replication environment, then these data dictionary views are not populated, and the RECOVER_OPERATION procedure cannot be used for the operation.

When the operation completes successfully, metadata about the operation is moved from the DBA_RECOVERABLE_SCRIPT view to the DBA_RECOVERABLE_SCRIPT_HIST view. The other views, DBA_RECOVERABLE_SCRIPT_PARAMS, DBA_RECOVERABLE_SCRIPT_BLOCKS, and DBA_RECOVERABLE_SCRIPT_ERRORS, retain information about the operation until it is purged automatically after 30 days.

When the operation encounters an error, you can use the RECOVER_OPERATION procedure in the DBMS_STREAMS_ADM package to either roll the operation forward, roll the operation back, or purge the metadata about the operation. Specifically, the operation_mode parameter in the RECOVER_OPERATION procedure provides the following options:

- FORWARD: This option attempts to complete the operation from the point at which it failed. Before specifying this option, correct the conditions that caused the errors reported in the DBA_RECOVERABLE_SCRIPT_ERRORS view.

  You can also use the FORWARD option to obtain more information about what caused the error. To do so, run SET SERVEROUTPUT ON in SQL*Plus and run the RECOVER_OPERATION procedure with the appropriate script ID. The RECOVER_OPERATION procedure shows the actions that lead to the error and the error numbers and messages.

- ROLLBACK: This option rolls back all of the actions performed by the operation. If the rollback is successful, then this option also moves the metadata about the operation from the DBA_RECOVERABLE_SCRIPT view to the DBA_RECOVERABLE_SCRIPT_HIST view. The other views retain information about the operation for 30 days.

- PURGE: This option moves the metadata about the operation from the DBA_RECOVERABLE_SCRIPT view to the DBA_RECOVERABLE_SCRIPT_HIST view

without rolling the operation back. The other views retain information about the operation for 30 days.

When a recovery operation is complete, information about the operation is stored in the DBA_RECOVERABLE_SCRIPT_HIST view. The STATUS column shows either EXECUTED or PURGED for each recovery operation.

Note: To run the RECOVER_OPERATION procedure, both databases must be Oracle Database 10*g* Release 2 or later databases.

See Also:

- "Configuring Replication Using the DBMS_STREAMS_ADM Package" on page 2-3 for more information about configuring an Oracle Streams replication environment with these procedures
- "Tasks to Complete Before Configuring Oracle Streams Replication" on page 1-13 for information about prerequisites that must be met before running these procedures
- *Oracle Database PL/SQL Packages and Types Reference*

Recovery Scenario

This section contains a scenario in which the MAINTAIN_SCHEMAS procedure stops because it encounters an error. Assume that the following procedure encountered an error when it was run at the capture database:

```
BEGIN
  DBMS_STREAMS_ADM.MAINTAIN_SCHEMAS(
    schema_names                 => 'hr',
    source_directory_object      => 'SOURCE_DIRECTORY',
    destination_directory_object => 'DEST_DIRECTORY',
    source_database              => 'dbs1.example.com',
    destination_database         => 'dbs2.example.com',
    perform_actions              => TRUE,
    dump_file_name               => 'export_hr.dmp',
    capture_queue_table          => 'rep_capture_queue_table',
    capture_queue_name           => 'rep_capture_queue',
    capture_queue_user           => NULL,
    apply_queue_table            => 'rep_dest_queue_table',
    apply_queue_name             => 'rep_dest_queue',
    apply_queue_user             => NULL,
    capture_name                 => 'capture_hr',
    propagation_name             => 'prop_hr',
    apply_name                   => 'apply_hr',
    log_file                     => 'export_hr.clg',
    bi_directional               => FALSE,
    include_ddl                  => TRUE,
    instantiation                => DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA);
END;
/
```

Complete the following steps to diagnose the problem and recover the operation:

1. In SQL*Plus, connect to the capture database as the Oracle Streams administrator.

   See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.

2. Determine the SCRIPT_ID value for the operation by running the following query:

```
SELECT SCRIPT_ID FROM DBA_RECOVERABLE_SCRIPT ORDER BY CREATION_TIME DESC;
```

   This query assumes that the most recent configuration operation is the one that encountered errors. Therefore, if more than one SCRIPT_ID is returned by the query, then use the first SCRIPT_ID in the list.

3. Query the DBA_RECOVERABLE_SCRIPT_ERRORS data dictionary view to determine the error and specify the SCRIPT_ID returned in Step 2 in the WHERE clause.

   For example, if the SCRIPT_ID is F73ED2C9E96B27B0E030578CB10B2424, then run the following query:

```
COLUMN SCRIPT_ID     HEADING 'Script ID'     FORMAT A35
COLUMN BLOCK_NUM     HEADING 'Block|Number' FORMAT 999999
COLUMN ERROR_MESSAGE HEADING 'Error Message' FORMAT A33

SELECT BLOCK_NUM, ERROR_MESSAGE
  FROM DBA_RECOVERABLE_SCRIPT_ERRORS
  WHERE SCRIPT_ID = 'F73ED2C9E96B27B0E030578CB10B2424';
```

   The query returns the following output:

```
Block
Number Error Message
------- --------------------------------
     12 ORA-39001: invalid argument value
```

4. Query the DBA_RECOVERABLE_SCRIPT_BLOCKS data dictionary view for the script ID returned in Step 2 and block number returned in Step 3 for information about the block in which the error occurred.

   For example, if the script ID is F73ED2C9E96B27B0E030578CB10B2424 and the block number is 12, run the following query:

```
COLUMN FORWARD_BLOCK        HEADING 'Forward Block'               FORMAT A50
COLUMN FORWARD_BLOCK_DBLINK HEADING 'Forward Block|Database Link' FORMAT A13
COLUMN STATUS               HEADING 'Status'                      FORMAT A12

SET LONG 10000
SELECT FORWARD_BLOCK,
       FORWARD_BLOCK_DBLINK,
       STATUS
  FROM DBA_RECOVERABLE_SCRIPT_BLOCKS
  WHERE SCRIPT_ID = 'F73ED2C9E96B27B0E030578CB10B2424' AND
        BLOCK_NUM = 12;
```

   The output contains the following information:

   - The FORWARD_BLOCK column contains detailed information about the actions performed by the procedure in the specified block. If necessary, spool the output into a file. In this scenario, the FORWARD_BLOCK column for block 12 contains the code for the Data Pump export.
   - The FORWARD_BLOCK_DBLINK column shows the database where the block is executed. In this scenario, the FORWARD_BLOCK_DBLINK column for block 12 shows DBS1.EXAMPLE.COM because the Data Pump export was being performed on DBS1.EXAMPLE.COM when the error occurred.

- The STATUS column shows the status of the block execution. In this scenario, the STATUS column for block 12 shows ERROR.

5. Optionally, run the RECOVER_OPERATION procedure operation at the capture database with SET SERVEROUTPUT ON to display more information about the errors:

```
SET SERVEROUTPUT ON
BEGIN
  DBMS_STREAMS_ADM.RECOVER_OPERATION(
    script_id       => 'F73ED2C9E96B27B0E030578CB10B2424',
    operation_mode  => 'FORWARD');
END;
/
```

With server output on, the actions that caused the error run again, and the actions and the resulting errors are displayed.

6. Interpret the output from the previous steps and diagnose the problem. The output returned in Step 3 provides the following information:

- The unique identifier for the configuration operation is F73ED2C9E96B27B0E030578CB10B2424. This value is the RAW value returned in the SCRIPT_ID field.
- Only one Oracle Streams configuration procedure is in the process of running because only one row was returned by the query. If multiple rows were returned by the query, then query the DBA_RECOVERABLE_SCRIPT and DBA_RECOVERABLE_SCRIPT_PARAMS views to determine which script ID applies to the configuration operation.
- The cause in *Oracle Database Error Messages* for the ORA-39001 error is the following: The user specified API parameters were of the wrong type or value range. Subsequent messages supplied by DBMS_DATAPUMP.GET_STATUS will further describe the error.
- The query on the DBA_RECOVERABLE_SCRIPT_BLOCKS view shows that the error occurred during Data Pump export.

The output from the queries shows that the MAINTAIN_SCHEMAS procedure encountered a Data Pump error. Notice that the instantiation parameter in the MAINTAIN_SCHEMAS procedure was set to DBMS_STREAMS_ADM.INSTANTIATION_SCHEMA. This setting means that the MAINTAIN_SCHEMAS procedure performs the instantiation using a Data Pump export and import. A Data Pump export dump file is generated to complete the export/import.

Data Pump errors usually are caused by one of the following conditions:

- One or more of the directory objects used to store the export dump file do not exist.
- The user running the procedure does not have access to specified directory objects.
- An export dump file with the same name as the one generated by the procedure already exists in a directory specified in the source_directory_object or destination_directory_object parameter.

7. Query the DBA_RECOVERABLE_SCRIPT_PARAMS data dictionary view at the capture database to determine the names of the directory objects specified when the MAINTAIN_SCHEMAS procedure was run:

```
COLUMN PARAMETER HEADING 'Parameter' FORMAT A30
COLUMN VALUE     HEADING 'Value'     FORMAT A45

SELECT PARAMETER,
       VALUE
       FROM DBA_RECOVERABLE_SCRIPT_PARAMS
       WHERE SCRIPT_ID = 'F73ED2C9E96B27B0E030578CB10B2424';
```

The query returns the following output:

```
Parameter                      Value
------------------------------ ---------------------------------------------
SOURCE_DIRECTORY_OBJECT        SOURCE_DIRECTORY
DESTINATION_DIRECTORY_OBJECT   DEST_DIRECTORY
SOURCE_DATABASE                DBS1.EXAMPLE
DESTINATION_DATABASE           DBS2.EXAMPLE
CAPTURE_QUEUE_TABLE            REP_CAPTURE_QUEUE_TABLE
CAPTURE_QUEUE_OWNER            STRMADMIN
CAPTURE_QUEUE_NAME             REP_CAPTURE_QUEUE
CAPTURE_QUEUE_USER
APPLY_QUEUE_TABLE              REP_DEST_QUEUE_TABLE
APPLY_QUEUE_OWNER              STRMADMIN
APPLY_QUEUE_NAME               REP_DEST_QUEUE
APPLY_QUEUE_USER
CAPTURE_NAME                   CAPTURE_HR
APPLY_NAME                     APPLY_HR
PROPAGATION_NAME               PROP_HR
INSTANTIATION                  INSTANTIATION_SCHEMA
BI_DIRECTIONAL                 TRUE
INCLUDE_DDL                    TRUE
LOG_FILE                       export_hr.clg
DUMP_FILE_NAME                 export_hr.dmp
SCHEMA_NAMES                   HR
```

8. Ensure that the directory object specified for the `source_directory_object` parameter exists at the source database, and ensure that the directory object specified for the `destination_directory_object` parameter exists at the destination database. Check for these directory objects by querying the `DBA_DIRECTORIES` data dictionary view.

   For this scenario, assume that the `SOURCE_DIRECTORY` directory object does not exist at the source database, and the `DEST_DIRECTORY` directory object does not exist at the destination database. The Data Pump error occurred because the directory objects used for the export dump file did not exist.

9. Create the required directory objects at the source and destination databases using the SQL statement `CREATE DIRECTORY`. See "Creating the Required Directory Objects" on page 2-15 for instructions.

10. Run the `RECOVER_OPERATION` procedure at the capture database:

```
BEGIN
  DBMS_STREAMS_ADM.RECOVER_OPERATION(
    script_id       => 'F73ED2C9E96B27B0E030578CB10B2424',
    operation_mode  => 'FORWARD');
END;
/
```

    Notice that the `script_id` parameter is set to the value determined in Step 3, and the `operation_mode` parameter is set to `FORWARD` to complete the configuration.

Also, the `RECOVER_OPERATION` procedure must be run at the database where the configuration procedure was run.

13

Comparing and Converging Data

This chapter contains instructions for comparing and converging data in database objects at two different databases using the DBMS_COMPARISON package. It also contains instructions for managing comparisons after they are created and for querying data dictionary views to obtain information about comparisons and comparison results.

This chapter contains these topics:

- About Comparing and Converging Data
- Other Documentation About the DBMS_COMPARISON Package
- Preparing To Compare and Converge a Shared Database Object
- Diverging a Database Object at Two Databases to Complete Examples
- Comparing a Shared Database Object at Two Databases
- Viewing Information About Comparisons and Comparison Results
- Converging a Shared Database Object
- Rechecking the Comparison Results for a Comparison
- Purging Comparison Results
- Dropping a Comparison
- Using DBMS_COMPARISON in an Oracle Streams Replication Environment

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

About Comparing and Converging Data

The DBMS_COMPARISON package enables you to compare database objects at different databases and identify differences in them. This package also enables you converge the database objects so that they are consistent at different databases. Typically, this package is used in environments that share a database object at multiple databases. When copies of the same database object exist at multiple databases, the database object is a **shared database object**.

Shared database objects might be maintained by data replication. For example, materialized views or Oracle Streams components might replicate the database objects and maintain them at multiple databases. A custom application might also maintain shared database objects. When a database object is shared, it can diverge at the databases that share it. You can use the DBMS_COMPARISON package to identify

differences in the shared database objects. After identifying the differences, you can optionally use this package to synchronize the shared database objects.

The DBMS_COMPARISON package can compare the following types of database objects:

- Tables
- Single-table views
- Materialized views
- Synonyms for tables, single-table views, and materialized views

Database objects of different types can be compared and converged at different databases. For example, a table at one database and a materialized view at another database can be compared and converged with this package.

You create a comparison between two database objects using the CREATE_COMPARISON procedure in the DBMS_COMPARISON package. After you create a comparison, you can run the comparison at any time using the COMPARE function. When you run the COMPARE function, it records **comparison results** in the appropriate data dictionary views. Separate comparison results are generated for each execution of the COMPARE function.

Scans

Each time the COMPARE function is run, one or more new **scan**s are performed for the specified comparison. A scan checks for differences in some or all of the rows in a shared database object at a single point in time. The comparison results for a single execution of the COMPARE function can include one or more scans. You can compare database objects multiple times, and a unique scan ID identifies each scan in the comparison results.

Buckets

A **bucket** is a range of rows in a database object that is being compared. Buckets improve performance by splitting the database object into ranges and comparing the ranges independently. Every comparison divides the rows being compared into an appropriate number of buckets. The number of buckets used depends on the size of the database object and is always less than the maximum number of buckets specified for the comparison by the max_num_buckets parameter in the CREATE_COMPARISON procedure.

When a bucket is compared using the COMPARE function, the following results are possible:

- No differences are found. In this case, the comparison proceeds to the next bucket.
- Differences are found. In this case, the comparison can split the bucket into smaller buckets and compare each smaller bucket. When differences are found in a smaller bucket, the bucket is split into still smaller buckets. This process continues until the minimum number of rows allowed in a bucket is reached. The minimum number of rows in a bucket for a comparison is specified by the min_rows_in_bucket parameter in the CREATE_COMPARISON procedure.

  When the minimum number of rows in a bucket is reached, the COMPARE function reports whether there are differences in the bucket. The COMPARE function includes the perform_row_dif parameter. This parameter controls whether the COMPARE function identifies each row difference in a bucket that has differences. When this parameter is set to TRUE, the COMPARE function identifies each row difference. When this parameter is set to FALSE, the COMPARE function does not

identify specific row differences. Instead, it only reports that there are differences in the bucket.

You can adjust the `max_num_buckets` and `min_rows_in_bucket` parameters in the `CREATE_COMPARISON` procedure to achieve the best performance when comparing a particular database object. After a comparison is created, you can view the bucket specifications for the comparison by querying the `MAX_NUM_BUCKETS` and `MIN_ROWS_IN_BUCKET` columns in the `DBA_COMPARISON` data dictionary view.

The `DBMS_COMPARISON` package uses the `ORA_HASH` function on the specified columns in all the rows in a bucket to compute a hash value for the bucket. If the hash values for two corresponding buckets match, then the contents of the buckets are assumed to match. The `ORA_HASH` function is an efficient way to compare buckets because row values are not transferred between databases. Instead, only the hash value is transferred.

Note: If an index column for a comparison is a `VARCHAR2` or `CHAR` column, then the number of buckets might exceed the value specified for the `max_num_buckets` parameter.

See Also:

- *Oracle Database SQL Language Reference* for more information about the `ORA_HASH` function
- *Oracle Database PL/SQL Packages and Types Reference* for information about index columns

Parent Scans and Root Scans

Each time the `COMPARE` function splits a bucket into smaller buckets, it performs new scans of the smaller buckets. The scan that analyzes a larger bucket is the **parent scan** of each scan that analyzes the smaller buckets into which the larger bucket was split. The **root scan** in the comparison results is the highest level parent scan. The root scan does not have a parent. You can identify parent and root scan IDs by querying the `DBA_COMPARISON_SCAN` data dictionary view.

You can recheck a scan using the `RECHECK` function, and you can converge a scan using the `CONVERGE` procedure. When you want to recheck or converge all of the rows in the comparison results, specify the root scan ID for the comparison results in the appropriate subprogram. When you want to recheck or converge a portion of the rows in comparison results, specify the scan ID of the scan that contains the differences.

For example, a scan with differences in 20 buckets is the parent scan for 20 additional scans, if each bucket with differences has more rows than the specified minimum number of rows in a bucket for the comparison. To view the minimum number of rows in a bucket for the comparison, query the `MIN_ROWS_IN_BUCKET` column in the `DBA_COMPARISON` data dictionary view.

See Also: *Oracle Database Reference* for information about the views related to the `DBMS_COMPARISON` package

How Scans and Buckets Identify Differences

This section describes two different comparison scenarios to show how scans and buckets identify differences in shared database objects. In each scenario, the `max_num_buckets` parameter is set to `3` in the `CREATE_COMPARISON` procedure.

Therefore, when the `COMPARE` or `RECHECK` function is run for the comparison, the comparison uses a maximum of three buckets in each scan.

Figure 13–1 shows the first scenario.

Figure 13–1 Comparison with max_num_buckets=3 and Differences in Each Bucket of Each Scan

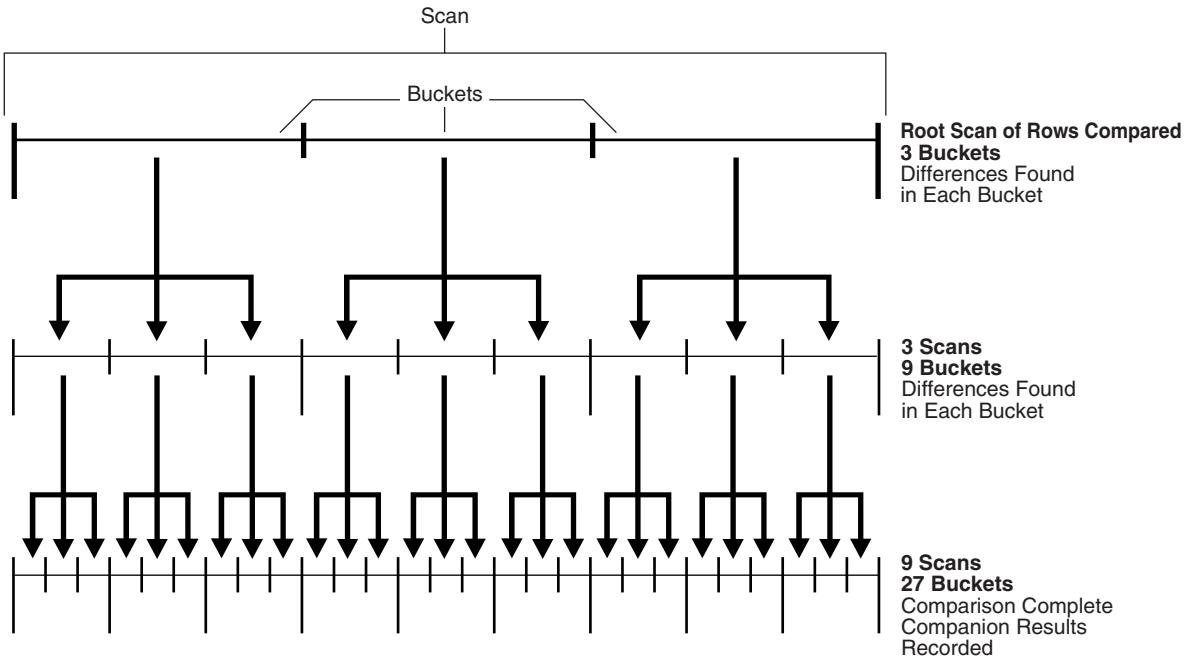


Figure 13–1 shows a line that represents the rows being compared in the shared database object. This figure illustrates how scans and buckets are used to identify differences when each bucket used by each scan has differences.

With the `max_num_buckets` parameter set to `3`, the comparison is executed in the following steps:

1. The root scan compares all of the rows in the current comparison. The root scan uses three buckets, and differences are found in each bucket.
2. A separate scan is performed on the rows in each bucket that was used by the root scan in the previous step. The current step uses three scans, and each scan uses three buckets. Therefore, this step uses a total of nine buckets. Differences are found in each bucket. In Figure 13–1, arrows show how each bucket from the root scan is split into three buckets for each of the scans in the current step.
3. A separate scan is performed on the rows in each bucket used by the scans in Step 2. This step uses nine scans, and each scan uses three buckets. Therefore, this step uses a total of 27 buckets. In Figure 13–1, arrows show how each bucket from Step 2 is split into three buckets for each of the scans in the current step.

After Step 3, the comparison results are recorded in the appropriate data dictionary views.

Figure 13–2 shows the second scenario.

Figure 13–2 Comparison with max_num_buckets=3 and Differences in One Bucket of Each Scan

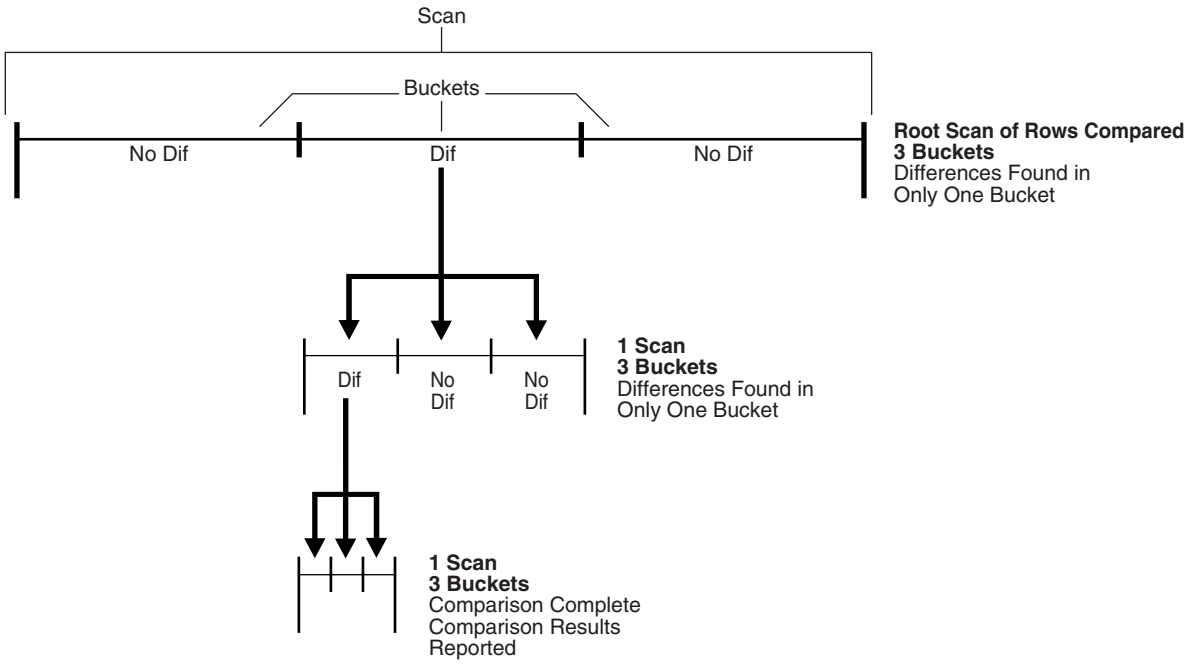


Figure 13–2 shows a line that represents the rows being compared in the shared database object. This figure illustrates how scans and buckets are used to identify differences when only one bucket used by each scan has differences.

With the `max_num_buckets` parameter set to `3`, the comparison is executed in the following steps:

1. The root scan compares all of the rows in the current comparison. The root scan uses three buckets, but differences are found in only one bucket.
2. A separate scan is performed on the rows in the one bucket that had differences. This step uses one scan, and the scan uses three buckets. Differences are found in only one bucket. In Figure 13–2, arrows show how the bucket with differences from the root scan is split into three buckets for the scan in the current step.
3. A separate scan is performed on the rows in the one bucket that had differences in Step 2. This step uses one scan, and the scan uses three buckets. In Figure 13–2, arrows show how the bucket with differences in Step 2 is split into three buckets for the scan in the current step.

After Step 3, the comparison results are recorded in the appropriate data dictionary views.

Note: This section describes scenarios in which the `max_num_buckets` parameter is set to `3` in the `CREATE_COMPARISON` procedure. This setting was chosen to illustrate how scans and buckets identify differences. Typically, the `max_num_buckets` parameter is set to a higher value. The default for this parameter is `1000`. You can adjust the parameter setting to achieve the best performance.

See Also:

- "Comparing a Shared Database Object at Two Databases" on page 13-7
- "Viewing Information About Comparisons and Comparison Results" on page 13-19

Other Documentation About the DBMS_COMPARISON Package

Please refer to the following documentation before completing the tasks described in this chapter:

- The *Oracle Database 2 Day + Data Replication and Integration Guide* contains basic information about the `DBMS_COMPARISON` package, including:
  - Basic conceptual information about the `DBMS_COMPARISON` package
  - Simple examples that describe using the package to compare and converge database objects
  - Sample queries that show information about the differences between database objects at different databases based on comparison results
- The chapter about the `DBMS_COMPARISON` package in the *Oracle Database PL/SQL Packages and Types Reference* contains advanced conceptual information about the package and detailed information about the subprograms in the package, including:
  - Requirements for using the package
  - Descriptions of constants used in the package
  - Descriptions of each subprogram in the package and its parameters

Preparing To Compare and Converge a Shared Database Object

Meet the following prerequisites before comparing and converging a shared database object at two databases:

- Configure network connectivity so that the two databases can communicate with each other. See *Oracle Database Net Services Administrator's Guide* for information about configuring network connectivity between databases.
- Identify or create a database user who will create, run, and manage comparisons. The database user must meet the privilege requirements described in the documentation for the `DBMS_COMPARISON` package in the *Oracle Database PL/SQL Packages and Types Reference*.

  After you identify or create a user with the required privileges, create a database link from the database that will run the subprograms in the `DBMS_COMPARISON` package to the other database that shares the database object. The identified user should own the database link, and the link should connect to a user with the required privileges on the remote database.

  For example, the following example creates a database link owned by a user named `admin` at the `comp1.example.com` database that connects to the `admin` user at the remote database `comp2.example.com`:

  1. In SQL*Plus, connect to the local database as `admin` user.

     See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.

2. Create the database link:

```
CREATE DATABASE LINK comp2.example.com CONNECT TO admin
   IDENTIFIED BY password USING 'comp2.example.com';
```

Diverging a Database Object at Two Databases to Complete Examples

The following sections contain examples that compare and converge a shared database object at two databases:

- "Comparing a Shared Database Object at Two Databases" on page 13-7
- "Converging a Shared Database Object" on page 13-30

Most of these examples compare and converge data in the `oe.orders` table. This table is part of the `oe` sample schema that is installed by default with Oracle Database. In these examples, the global names of the databases are `comp1.example.com` and `comp2.example.com`, but you can substitute any two databases in your environment that meet the prerequisites described in "Preparing To Compare and Converge a Shared Database Object" on page 13-6.

For the purposes of the examples, make the `oe.orders` table diverge at two databases by completing the following steps:

1. In SQL*Plus, connect to the `comp2.example.com` database as `oe` user.

   See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.

2. Delete the orders in the `oe.orders` table with a `customer_id` equal to `147`:

   ```
   DELETE FROM oe.orders WHERE customer_id=147;
   ```

3. Modify the data in a row in the `oe.orders` table:

   ```
   UPDATE oe.orders SET sales_rep_id=163 WHERE order_id=2440;
   ```

4. Insert a row into the `oe.orders` table:

   ```
   INSERT INTO oe.orders VALUES(3000, TIMESTAMP '2006-01-01 2:00:00', 'direct',
   107, 3, 16285.21, 156, NULL);
   ```

5. Commit your changes and exit SQL*Plus:

   ```
   COMMIT;
   EXIT
   ```

Note: Usually, these steps are not required. They are included to ensure that the `oe.orders` table diverges at the two databases.

Comparing a Shared Database Object at Two Databases

The examples in this section use the `DBMS_COMPARISON` package to compare the `oe.orders` table at the `comp1.example.com` and `comp2.example.com` databases. The examples use the package to create different types of comparisons and compare the tables with the comparisons.

This section contains the following examples:

- Comparing a Subset of Columns in a Shared Database Object
- Comparing a Shared Database Object without Identifying Row Differences
- Comparing a Random Portion of a Shared Database Object
- Comparing a Shared Database Object Cyclically
- Comparing a Custom Portion of a Shared Database Object
- Comparing a Shared Database Object That Contains CLOB or BLOB Columns

Comparing a Subset of Columns in a Shared Database Object

The `column_list` parameter in the `CREATE_COMPARISON` procedure enables you to compare a subset of the columns in a database object. The following are reasons to compare a subset of columns:

- A database object contains extra columns that do not exist in the database object to which it is being compared. In this case, the `column_list` parameter must only contain the columns that exist in both database objects.
- You want to focus a comparison on a specific set of columns. For example, if a table contains hundreds of columns, then you might want to list specific columns in the `column_list` parameter to make the comparison more efficient.
- Differences are expected in some columns. In this case, exclude the columns in which differences are expected from the `column_list` parameter.

The columns in the column list must meet the following requirements:

- The column list must meet the index column requirements for the `DBMS_COMPARISON` package. See *Oracle Database PL/SQL Packages and Types Reference* for information about index column requirements.
- If you plan to use the `CONVERGE` procedure to make changes to a database object based on comparison results, then you must include in the column list any column in this database object that has a `NOT NULL` constraint but no default value.

This example compares the `order_id`, `order_date`, and `customer_id` columns in the `oe.orders` table at the `comp1.example.com` and `comp2.example.com` databases:

1. Complete the tasks described in "Preparing To Compare and Converge a Shared Database Object" on page 13-6 and "Diverging a Database Object at Two Databases to Complete Examples" on page 13-7.
2. In SQL*Plus, connect to the `comp1.example.com` database as the administrative user who owns the database link created in "Preparing To Compare and Converge a Shared Database Object" on page 13-6.

   See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.
3. Run the `CREATE_COMPARISON` procedure to create the comparison:

```
BEGIN
  DBMS_COMPARISON.CREATE_COMPARISON(
    comparison_name => 'compare_subset_columns',
    schema_name     => 'oe',
    object_name     => 'orders',
    dblink_name     => 'comp2.example.com',
    column_list     => 'order_id,order_date,customer_id');
END;
```

```
/
```

Note that the name of the new comparison is `compare_subset_columns`. This comparison is owned by the user who runs the `CREATE_COMPARISON` procedure.

4. Run the `COMPARE` function to compare the `oe.orders` table at the two databases:

```
SET SERVEROUTPUT ON
DECLARE
  consistent   BOOLEAN;
  scan_info    DBMS_COMPARISON.COMPARISON_TYPE;
BEGIN
  consistent := DBMS_COMPARISON.COMPARE(
                  comparison_name => 'compare_subset_columns',
                  scan_info       => scan_info,
                  perform_row_dif => TRUE);
  DBMS_OUTPUT.PUT_LINE('Scan ID: '||scan_info.scan_id);
  IF consistent=TRUE THEN
    DBMS_OUTPUT.PUT_LINE('No differences were found.');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Differences were found.');
  END IF;
END;
/
```

Notice that the `perform_row_dif` parameter is set to `TRUE` in the `COMPARE` function. This setting instructs the `COMPARE` function to identify each individual row difference in the tables. When the `perform_row_dif` parameter is set to `FALSE`, the `COMPARE` function records whether there are differences in the tables, but does not record each individual row difference.

Your output is similar to the following:

```
Scan ID: 1
Differences were found.

PL/SQL procedure successfully completed.
```

See Also:

- "Viewing Detailed Information About the Row Differences Found in a Scan" on page 13-27
- "Converging a Shared Database Object" on page 13-30 to converge the differences found in the comparison results
- "Rechecking the Comparison Results for a Comparison" on page 13-34 to recheck the comparison results

Comparing a Shared Database Object without Identifying Row Differences

When you run the `COMPARE` procedure for an existing comparison, the `perform_row_dif` parameter controls whether the `COMPARE` procedure identifies each individual row difference in the database objects:

- When the `perform_row_dif` parameter is set to `TRUE`, the `COMPARE` procedure records whether there are differences in the database objects, and it records each individual row difference. Set this parameter to `TRUE` when you must identify each difference in the database objects.

- When the `perform_row_dif` parameter is set to `FALSE`, the `COMPARE` procedure records whether there are differences in the database objects, but does not record each individual row difference. Set this parameter to `FALSE` when you want to know if there are differences in the database objects, but you do not need to identify each individual difference. Setting this parameter to `FALSE` is the most efficient way to perform a comparison.

See *Oracle Database PL/SQL Packages and Types Reference* for information about the `perform_row_dif` parameter in the `COMPARE` function.

This example compares the entire `oe.orders` table at the `comp1.example.com` and `comp2.example.com` databases without identifying individual row differences:

1. Complete the tasks described in "Preparing To Compare and Converge a Shared Database Object" on page 13-6 and "Diverging a Database Object at Two Databases to Complete Examples" on page 13-7.

2. In SQL*Plus, connect to the `comp1.example.com` database as the administrative user who owns the database link created in "Preparing To Compare and Converge a Shared Database Object" on page 13-6.

   See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.

3. Run the `CREATE_COMPARISON` procedure to create the comparison:

```
BEGIN
  DBMS_COMPARISON.CREATE_COMPARISON(
    comparison_name => 'compare_orders',
    schema_name     => 'oe',
    object_name     => 'orders',
    dblink_name     => 'comp2.example.com');
END;
/
```

4. Run the `COMPARE` function to compare the `oe.orders` table at the two databases:

```
SET SERVEROUTPUT ON
DECLARE
  consistent   BOOLEAN;
  scan_info    DBMS_COMPARISON.COMPARISON_TYPE;
BEGIN
  consistent := DBMS_COMPARISON.COMPARE(
                  comparison_name => 'compare_orders',
                  scan_info       => scan_info,
                  perform_row_dif => FALSE);
  DBMS_OUTPUT.PUT_LINE('Scan ID: '||scan_info.scan_id);
  IF consistent=TRUE THEN
    DBMS_OUTPUT.PUT_LINE('No differences were found.');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Differences were found.');
  END IF;
END;
/
```

   Notice that the `perform_row_dif` parameter is set to `FALSE` in the `COMPARE` function.

   Your output is similar to the following:

```
Scan ID: 4
Differences were found.
```

```
PL/SQL procedure successfully completed.
```

See Also:

- "Viewing Detailed Information About the Row Differences Found in a Scan" on page 13-27
- "Converging a Shared Database Object" on page 13-30 to converge the differences found in the comparison results
- "Rechecking the Comparison Results for a Comparison" on page 13-34 to recheck the comparison results

Comparing a Random Portion of a Shared Database Object

The `scan_percent` and `scan_mode` parameters in the `CREATE_COMPARISON` procedure enable you to compare a random portion of a shared database object instead of the entire database object. Typically, you use this option under the following conditions:

- You are comparing a relatively large shared database object, and you want to determine whether there might be differences without devoting the resources and time to comparing the entire database object.
- You do not intend to use subsequent comparisons to compare different portions of the database object. If you want to compare different portions of the database object in subsequent comparisons, then see "Comparing a Shared Database Object Cyclically" on page 13-12 for instructions.

This example compares a random portion of the `oe.orders` table at the `comp1.example.com` and `comp2.example.com` databases:

1. Complete the tasks described in "Preparing To Compare and Converge a Shared Database Object" on page 13-6 and "Diverging a Database Object at Two Databases to Complete Examples" on page 13-7.
2. In SQL*Plus, connect to the `comp1.example.com` database as the administrative user who owns the database link created in "Preparing To Compare and Converge a Shared Database Object" on page 13-6.

   See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.
3. Run the `CREATE_COMPARISON` procedure to create the comparison:

   ```
   BEGIN
     DBMS_COMPARISON.CREATE_COMPARISON(
       comparison_name => 'compare_random',
       schema_name     => 'oe',
       object_name     => 'orders',
       dblink_name     => 'comp2.example.com',
       scan_mode       =>  DBMS_COMPARISON.CMP_SCAN_MODE_RANDOM,
       scan_percent    =>  50);
   END;
   /
   ```

   Notice that the `scan_percent` parameter is set to `50` to specify that the comparison scans half of the table. The `scan_mode` parameter is set to `DBMS_COMPARISON.CMP_SCAN_MODE_RANDOM` to specify that the comparison compares random rows in the table.
4. Run the `COMPARE` function to compare the `oe.orders` table at the two databases:

```
SET SERVEROUTPUT ON
DECLARE
  consistent   BOOLEAN;
  scan_info    DBMS_COMPARISON.COMPARISON_TYPE;
BEGIN
  consistent := DBMS_COMPARISON.COMPARE(
                  comparison_name => 'compare_random',
                  scan_info       => scan_info,
                  perform_row_dif => TRUE);
  DBMS_OUTPUT.PUT_LINE('Scan ID: '||scan_info.scan_id);
  IF consistent=TRUE THEN
    DBMS_OUTPUT.PUT_LINE('No differences were found.');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Differences were found.');
  END IF;
END;
/
```

Notice that the `perform_row_dif` parameter is set to `TRUE` in the `COMPARE` function. This setting instructs the `COMPARE` function to identify each individual row difference in the tables. When the `perform_row_dif` parameter is set to `FALSE`, the `COMPARE` function records whether there are differences in the tables, but does not record each individual row difference.

Your output is similar to the following:

```
Scan ID: 7
Differences were found.

PL/SQL procedure successfully completed.
```

This comparison scan might or might not find differences, depending on the portion of the table that is compared.

See Also:

- "Viewing Detailed Information About the Row Differences Found in a Scan" on page 13-27
- "Converging a Shared Database Object" on page 13-30 to converge the differences found in the comparison results
- "Rechecking the Comparison Results for a Comparison" on page 13-34 to recheck the comparison results

Comparing a Shared Database Object Cyclically

The `scan_percent` and `scan_mode` parameters in the `CREATE_COMPARISON` procedure enable you to compare a portion of a shared database object cyclically. A cyclic comparison scans a portion of the database object being compared during a single comparison. When the database object is compared again, another portion of the database object is compared, starting where the last comparison ended.

Typically, you use this option under the following conditions:

- You are comparing a relatively large shared database object, and you want to determine whether there might be differences without devoting the resources and time to comparing the entire database object.
- You want each comparison to compare a different portion of the shared database object, so that the entire database object is compared with the appropriate number

of scans. For example, if you compare 25% of the shared database object, then the entire database object is compared after four comparisons. If you do not want to compare different portions of the database object in subsequent comparisons, see "Comparing a Random Portion of a Shared Database Object" on page 13-11 for instructions.

This example compares `oe.orders` table cyclically at the `comp1.example.com` and `comp2.example.com` databases:

1. Complete the tasks described in "Preparing To Compare and Converge a Shared Database Object" on page 13-6 and "Diverging a Database Object at Two Databases to Complete Examples" on page 13-7.

2. In SQL*Plus, connect to the `comp1.example.com` database as the administrative user who owns the database link created in "Preparing To Compare and Converge a Shared Database Object" on page 13-6.

   See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.

3. Run the `CREATE_COMPARISON` procedure to create the comparison:

```
BEGIN
  DBMS_COMPARISON.CREATE_COMPARISON(
    comparison_name => 'compare_cyclic',
    schema_name     => 'oe',
    object_name     => 'orders',
    dblink_name     => 'comp2.example.com',
    scan_mode       =>  DBMS_COMPARISON.CMP_SCAN_MODE_CYCLIC,
    scan_percent    =>  50);
END;
/
```

   Notice that the `scan_percent` parameter is set to `50` to specify that the comparison scans half of the table. The `scan_mode` parameter is set to `DBMS_COMPARISON.CMP_SCAN_MODE_CYCLIC` to specify that the comparison compares rows in the table cyclically.

4. Run the `COMPARE` function to compare the `oe.orders` table at the two databases:

```
SET SERVEROUTPUT ON
DECLARE
  consistent   BOOLEAN;
  scan_info    DBMS_COMPARISON.COMPARISON_TYPE;
BEGIN
  consistent := DBMS_COMPARISON.COMPARE(
                  comparison_name => 'compare_cyclic',
                  scan_info       => scan_info,
                  perform_row_dif => TRUE);
  DBMS_OUTPUT.PUT_LINE('Scan ID: '||scan_info.scan_id);
  IF consistent=TRUE THEN
    DBMS_OUTPUT.PUT_LINE('No differences were found.');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Differences were found.');
  END IF;
END;
/
```

   Notice that the `perform_row_dif` parameter is set to `TRUE` in the `COMPARE` function. This setting instructs the `COMPARE` function to identify each individual row difference in the tables. When the `perform_row_dif` parameter is set to

FALSE, the COMPARE function records whether there are differences in the tables, but does not record each individual row difference.

Your output is similar to the following:

```
Scan ID: 8
Differences were found.

PL/SQL procedure successfully completed.
```

This comparison scan might or might not find differences, depending on the portion of the table that is compared.

5. To compare the next portion of the database object, starting where the last comparison ended, rerun the COMPARE function that was run in Step 4. In this example, running the COMPARE function twice compares the entire database object because the scan_percent parameter was set to 50 in Step 3.

See Also:

- "Viewing Detailed Information About the Row Differences Found in a Scan" on page 13-27
- "Converging a Shared Database Object" on page 13-30 to converge the differences found in the comparison results
- "Rechecking the Comparison Results for a Comparison" on page 13-34 to recheck the comparison results

Comparing a Custom Portion of a Shared Database Object

The scan_mode parameter in the CREATE_COMPARISON procedure enables you to compare a custom portion of a shared database object. After a comparison is created with the scan_mode parameter set to CMP_SCAN_MODE_CUSTOM in the CREATE_COMPARISON procedure, you can specify the exact portion of the database object to compare when you run the COMPARE function.

Typically, you use this option under the following conditions:

- You have a specific portion of a shared database object that you want to compare.
- You are comparing a relatively large shared database object, and you want to determine whether there might be difference in a specific portion of it without devoting the resources and time to comparing the entire database object.

See *Oracle Database PL/SQL Packages and Types Reference* for information about the scan_mode parameter in the CREATE_COMPARISON procedure.

This example compares a custom portion of the oe.orders table at the comp1.example.com and comp2.example.com databases:

1. Complete the tasks described in "Preparing To Compare and Converge a Shared Database Object" on page 13-6 and "Diverging a Database Object at Two Databases to Complete Examples" on page 13-7.
2. In SQL*Plus, connect to the comp1.example.com database as the administrative user who owns the database link created in "Preparing To Compare and Converge a Shared Database Object" on page 13-6.

   See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.
3. Run the CREATE_COMPARISON procedure to create the comparison:

```
BEGIN
  DBMS_COMPARISON.CREATE_COMPARISON(
    comparison_name   => 'compare_custom',
    schema_name       => 'oe',
    object_name       => 'orders',
    dblink_name       => 'comp2.example.com',
    index_schema_name => 'oe',
    index_name        => 'order_pk',
    scan_mode         =>  DBMS_COMPARISON.CMP_SCAN_MODE_CUSTOM);
END;
/
```

Notice that the `scan_mode` parameter is set to `DBMS_COMPARISON.CMP_SCAN_MODE_CUSTOM`. When you specify this scan mode, you should specify the index to use for the comparison. This example specifies the `or.order_pk` index.

4. Identify the index column or columns for the comparison created in Step 3 by running the following query:

```
SELECT COLUMN_NAME, COLUMN_POSITION FROM DBA_COMPARISON_COLUMNS
  WHERE COMPARISON_NAME = 'COMPARE_CUSTOM' AND
        INDEX_COLUMN    = 'Y';
```

For a custom comparison, you use the index column to specify the portion of the table to compare when you run the `COMPARE` function in the next step. In this example, the query should return the following output:

```
COLUMN_NAME                    COLUMN_POSITION
------------------------------ ---------------
ORDER_ID                                     1
```

This output shows that the `order_id` column in the `oe.orders` table is the index column for the comparison.

For other database objects, the `CREATE_COMPARISON` procedure might identify multiple index columns. If there are multiple index columns, then specify values for the lead index column in the next step. The lead index column shows `1` for its `COLUMN_POSITION` value.

5. Run the `COMPARE` function to compare the `oe.orders` table at the two databases:

```
SET SERVEROUTPUT ON
DECLARE
  consistent   BOOLEAN;
  scan_info    DBMS_COMPARISON.COMPARISON_TYPE;
BEGIN
  consistent := DBMS_COMPARISON.COMPARE(
                  comparison_name => 'compare_custom',
                  scan_info       => scan_info,
                  min_value       => '2430',
                  max_value       => '2460',
                  perform_row_dif => TRUE);
  DBMS_OUTPUT.PUT_LINE('Scan ID: '||scan_info.scan_id);
  IF consistent=TRUE THEN
    DBMS_OUTPUT.PUT_LINE('No differences were found.');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Differences were found.');
  END IF;
END;
/
```

Notice the following parameter settings in the `COMPARE` function:

- The `min_value` and `max_value` parameters are set to `2430` and `2460`, respectively. Therefore, the `COMPARE` function only compares the range of rows that begins with `2430` and ends with `2460` in the `order_id` column.
- The `min_value` and `max_value` parameters are specified as `VARCHAR2` data type values, even though the column data type for the `order_id` column is `NUMBER`.
- The `perform_row_dif` parameter is set to `TRUE` in the `COMPARE` function. This setting instructs the `COMPARE` function to identify each individual row difference in the tables. When the `perform_row_dif` parameter is set to `FALSE`, the `COMPARE` function records whether there are differences in the tables, but does not record each individual row difference.

Your output is similar to the following:

```
Scan ID: 10
Differences were found.

PL/SQL procedure successfully completed.
```

See Also:

- "Viewing Detailed Information About the Row Differences Found in a Scan" on page 13-27
- "Converging a Shared Database Object" on page 13-30 to converge the differences found in the comparison results
- "Rechecking the Comparison Results for a Comparison" on page 13-34 to recheck the comparison results

Comparing a Shared Database Object That Contains CLOB or BLOB Columns

The `DBMS_COMPARISON` package does not support directly comparing a shared database object that contains a column of either `CLOB` or `BLOB` data type. However, you can complete these basic steps to compare a table with a `CLOB` or `BLOB` column:

1. At each database, create a view based on the table and replace the `CLOB` or `BLOB` column with a `RAW` data type column that is generated using the `DBMS_CRYPTO.HASH` function.
2. Compare the views created in Step 1.

The illustrates how complete these steps for a simple table with a `NUMBER` column and a `CLOB` column. In this example, the global names of the databases are `comp1.example.com` and `comp2.example.com`, but you can substitute any two databases in your environment that meet the prerequisites described in "Preparing To Compare and Converge a Shared Database Object" on page 13-6.

Note: The `DBMS_COMPARISON` package cannot converge a shared database object that contains LOB columns.

Complete the following steps:

1. Complete the tasks described in "Preparing To Compare and Converge a Shared Database Object" on page 13-6.

2. At the `comp1.example.com` database, ensure that the user who owns or will own the table with the `CLOB` or `BLOB` column has `EXECUTE` privilege on the `DBMS_CRYPTO` package.

   In this example, assume the user who will own the table is `oe`. Complete the following steps to grant this privilege to `oe` user:

   a. In SQL*Plus, connect to the `comp1.example.com` database as an administrative user who can grant privileges.

   b. Grant `EXECUTE` on the `DBMS_CRYPTO` package to the user:

   ```
   GRANT EXECUTE ON DBMS_CRYPTO TO oe;
   ```

3. At the `comp2.example.com` database, ensure that the user who owns or will own the table with the `CLOB` or `BLOB` column has `EXECUTE` privilege on the `DBMS_CRYPTO` package.

   In this example, assume the user who will own the table is `oe`. Complete the following steps to grant this privilege to `oe` user:

   a. In SQL*Plus, connect to the `comp2.example.com` database as an administrative user who can grant privileges.

   b. Grant `EXECUTE` on the `DBMS_CRYPTO` package to the user:

   ```
   GRANT EXECUTE ON DBMS_CRYPTO TO oe;
   ```

4. Create the table with the `CLOB` column and the view based on the table in the `comp1.example.com` database:

   a. In SQL*Plus, connect to the `comp1.example.com` database as the user who will own the table.

      See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.

   b. Create the table:

   ```
   CREATE TABLE oe.tab_lob(
     c1 NUMBER PRIMARY KEY,
     c2 CLOB DEFAULT to_clob('c2'));
   ```

   c. Insert a row into the `tab_lob` table and commit the change:

   ```
   INSERT INTO oe.tab_lob VALUES(1, TO_CLOB('row 1'));
   COMMIT;
   ```

   d. Create the view:

   ```
   BEGIN
     EXECUTE IMMEDIATE 'CREATE VIEW view_lob AS SELECT
         c1,
         DBMS_CRYPTO.HASH(c2, '||DBMS_CRYPTO.HASH_SH1||') c2_hash
       FROM tab_lob';
   END;
   /
   ```

   See Also: *Oracle Database PL/SQL Packages and Types Reference* for more information about the cryptographic hash functions used in the `DBMS_CRYPTO` package

5. Create the table with the CLOB column and the view based on the table in the comp2.example.com database:

   a. In SQL*Plus, connect to the comp2.example.com database as the user who will own the table.

      See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.

   b. Create the table:

```
CREATE TABLE oe.tab_lob(
  c1 NUMBER PRIMARY KEY,
  c2 CLOB DEFAULT to_clob('c2'));
```

   c. Insert a row into the tab_lob table and commit the change:

```
INSERT INTO oe.tab_lob VALUES(1, TO_CLOB('row 1'));
COMMIT;
```

   d. Create the view:

```
BEGIN
  EXECUTE IMMEDIATE 'CREATE VIEW view_lob AS SELECT
      c1,
      DBMS_CRYPTO.HASH(c2, '||DBMS_CRYPTO.HASH_SH1||') c2_hash
    FROM tab_lob';
END;
/
```

6. In SQL*Plus, connect to the comp1.example.com database as the administrative user who owns the database link created in "Preparing To Compare and Converge a Shared Database Object" on page 13-6.

7. Run the CREATE_COMPARISON procedure to create the comparison:

```
BEGIN
  DBMS_COMPARISON.CREATE_COMPARISON(
    comparison_name => 'compare_lob',
    schema_name     => 'oe',
    object_name     => 'view_lob',
    dblink_name     => 'comp2.example.com');
END;
/
```

   Notice that the schema_name and object_name parameters specify the view oe.view_lob and not the table that contains the CLOB column.

8. Run the COMPARE function to compare the oe.view_lob view at the two databases:

```
SET SERVEROUTPUT ON
DECLARE
  consistent   BOOLEAN;
  scan_info    DBMS_COMPARISON.COMPARISON_TYPE;
BEGIN
  consistent := DBMS_COMPARISON.COMPARE(
                  comparison_name => 'compare_lob',
                  scan_info       => scan_info,
                  perform_row_dif => TRUE);
  DBMS_OUTPUT.PUT_LINE('Scan ID: '||scan_info.scan_id);
  IF consistent=TRUE THEN
    DBMS_OUTPUT.PUT_LINE('No differences were found.');
```

```
  ELSE
    DBMS_OUTPUT.PUT_LINE('Differences were found.');
  END IF;
END;
/

Scan ID: 1
No differences were found.

PL/SQL procedure successfully completed.
```

9. Make the `oe.tab_lob` table diverge at two databases by completing the following steps:

   a. In SQL*Plus, connect to the `comp1.example.com` database as the user who owns the table.

      See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.

   b. Insert a row and commit the change:

      ```
      INSERT INTO oe.tab_lob VALUES(2, TO_CLOB('row a'));
      COMMIT;
      ```

   c. In SQL*Plus, connect to the `comp2.example.com` database as the user who owns the table.

      See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.

   d. Insert a row and commit the change:

      ```
      INSERT INTO oe.tab_lob VALUES(2, TO_CLOB('row b'));
      COMMIT;
      ```

10. Run the `COMPARE` function again to compare the `oe.view_lob` view at the two databases. See Step 8.

    The shared table with the `CLOB` column has diverged at the two databases. Therefore, when you compare the view, the `COMPARE` function returns the following output:

    ```
    Scan ID: 2
    Differences were found.

    PL/SQL procedure successfully completed.
    ```

Viewing Information About Comparisons and Comparison Results

The following data dictionary views contain information about comparisons created with the `DBMS_COMPARISON` package:

- `DBA_COMPARISON`
- `USER_COMPARISON`
- `DBA_COMPARISON_COLUMNS`
- `USER_COMPARISON_COLUMNS`
- `DBA_COMPARISON_SCAN`

- USER_COMPARISON_SCAN
- DBA_COMPARISON_SCAN_VALUES
- USER_COMPARISON_SCAN_VALUES
- DBA_COMPARISON_ROW_DIF
- USER_COMPARISON_ROW_DIF

The following sections contain sample queries that you can use to monitor comparisons and comparison results:

- Viewing General Information About the Comparisons in a Database
- Viewing Information Specific to Random and Cyclic Comparisons
- Viewing the Columns Compared by Each Comparison in a Database
- Viewing General Information About Each Scan in a Database
- Viewing the Parent Scan ID and Root Scan ID for Each Scan in a Database
- Viewing Detailed Information About the Row Differences Found in a Scan
- Viewing Information About the Rows Compared in Specific Scans

> **See Also:** *Oracle Database Reference* for detailed information about the data dictionary views related to comparisons

Viewing General Information About the Comparisons in a Database

The DBA_COMPARISON data dictionary view contains information about the comparisons in the local database. The query in this section displays the following information about each comparison:

- The owner of the comparison
- The name of the comparison
- The schema that contains the database object compared by the comparison
- The name of the database object compared by the comparison
- The data type of the database object compared by the comparison
- The scan mode used by the comparison. The following scan modes are possible:
  - FULL indicates that the entire database object is compared.
  - RANDOM indicates that a random portion of the database object is compared.
  - CYCLIC indicates that a portion of the database object is compared during a single comparison. When the database object is compared again, another portion of the database object is compared, starting where the last compare ended.
  - CUSTOM indicates that the COMPARE function specifies the range to compare in the database object.
- The name of the database link used to connect with the remote database

To view this information, run the following query:

```
COLUMN OWNER HEADING 'Comparison|Owner' FORMAT A10
COLUMN COMPARISON_NAME HEADING 'Comparison|Name' FORMAT A22
COLUMN SCHEMA_NAME HEADING 'Schema|Name' FORMAT A8
COLUMN OBJECT_NAME HEADING 'Object|Name' FORMAT A8
```

```
COLUMN OBJECT_TYPE HEADING 'Object|Type' FORMAT A8
COLUMN SCAN_MODE HEADING 'Scan|Mode' FORMAT A6
COLUMN DBLINK_NAME HEADING 'Database|Link' FORMAT A15

SELECT OWNER,
       COMPARISON_NAME,
       SCHEMA_NAME,
       OBJECT_NAME,
       OBJECT_TYPE,
       SCAN_MODE,
       DBLINK_NAME
  FROM DBA_COMPARISON;
```

Your output is similar to the following:

```
Comparison Comparison             Schema   Object   Object   Scan   Database
Owner      Name                   Name     Name     Type     Mode   Link
---------- ---------------------- -------- -------- -------- ------ ----------
ADMIN      COMPARE_SUBSET_COLUMNS OE       ORDERS   TABLE    FULL   COMP2.EXAM
                                                                     PLE
ADMIN      COMPARE_ORDERS         OE       ORDERS   TABLE    FULL   COMP2.EXAM
                                                                     PLE
ADMIN      COMPARE_RANDOM         OE       ORDERS   TABLE    RANDOM COMP2.EXAM
                                                                     PLE
ADMIN      COMPARE_CYCLIC         OE       ORDERS   TABLE    CYCLIC COMP2.EXAM
                                                                     PLE
ADMIN      COMPARE_CUSTOM         OE       ORDERS   TABLE    CUSTOM COMP2.EXAM
                                                                     PLE
```

A comparison compares the local database object with a database object at a remote database. The comparison uses the database link shown by the query to connect to the remote database and perform the comparison.

By default, a comparison assumes that the owner, name, and data type of the database objects being compared are the same at both databases. However, they can be different at the local and remote databases. The query in this section does not display information about the remote database object, but you can query the `REMOTE_SCHEMA_NAME`, `REMOTE_OBJECT_NAME`, and `REMOTE_OBJECT_TYPE` columns to view this information.

> **See Also:** Comparing a Shared Database Object at Two Databases on page 13-7 for information about creating the comparisons shown in the output of this query

Viewing Information Specific to Random and Cyclic Comparisons

When you create comparisons that use the scan modes `RANDOM` or `CYCLIC`, you specify the percentage of the shared database object to compare. The query in this section shows the following information about random and cyclic comparisons:

- The owner of the comparison
- The name of the comparison
- The schema that contains the database object compared by the comparison
- The name of the database object compared by the comparison
- The data type of the database object compared by the comparison

- The scan percentage for the comparison. Each time the COMPARE function is run to perform a comparison scan, the specified percentage of the database object is compared.
- The last lead index column value used by the comparison. The next time the COMPARE function is run, it will start with row that has a lead index column value that directly follows the value shown by the query. This value only applies to cyclic comparisons.

To view this information, run the following query:

```
COLUMN OWNER HEADING 'Comparison|Owner' FORMAT A10
COLUMN COMPARISON_NAME HEADING 'Comparison|Name' FORMAT A22
COLUMN SCHEMA_NAME HEADING 'Schema|Name' FORMAT A8
COLUMN OBJECT_NAME HEADING 'Object|Name' FORMAT A8
COLUMN OBJECT_TYPE HEADING 'Object|Type' FORMAT A8
COLUMN SCAN_PERCENT HEADING 'Scan|Percent' FORMAT 999
COLUMN CYCLIC_INDEX_VALUE HEADING 'Cyclic|Index|Value' FORMAT A10

SELECT OWNER,
       COMPARISON_NAME,
       SCHEMA_NAME,
       OBJECT_NAME,
       OBJECT_TYPE,
       SCAN_PERCENT,
       CYCLIC_INDEX_VALUE
  FROM DBA_COMPARISON
  WHERE SCAN_PERCENT IS NOT NULL;
```

Your output is similar to the following:

```
                                                                  Cyclic
Comparison Comparison             Schema   Object   Object      Scan Index
Owner      Name                   Name     Name     Type     Percent Value
---------- ---------------------- -------- -------- -------- ------- ----------
ADMIN      COMPARE_RANDOM         OE       ORDERS   TABLE         50
ADMIN      COMPARE_CYCLIC         OE       ORDERS   TABLE         50 2677
```

See Also:

- "Comparing a Random Portion of a Shared Database Object" on page 13-11
- "Comparing a Shared Database Object Cyclically" on page 13-12
- "Viewing General Information About the Comparisons in a Database" on page 13-20

Viewing the Columns Compared by Each Comparison in a Database

When you create a comparison, you can specify that the comparison compares all of the columns in the shared database object or a subset of the columns. Also, you can specify an index for the comparison to use or let the system identify an index automatically.

The query in this section displays the following information:

- The owner of the comparison
- The name of the comparison
- The schema that contains the database object compared by the comparison

- The name of the database object compared by the comparison
- The column name of each column being compared in each database object
- The column position of each column
- Whether a column is an index column

To display this information, run the following query:

```
COLUMN OWNER HEADING 'Comparison|Owner' FORMAT A10
COLUMN COMPARISON_NAME HEADING 'Comparison|Name' FORMAT A15
COLUMN SCHEMA_NAME HEADING 'Schema|Name' FORMAT A10
COLUMN OBJECT_NAME HEADING 'Object|Name' FORMAT A10
COLUMN COLUMN_NAME HEADING 'Column|Name' FORMAT A12
COLUMN COLUMN_POSITION HEADING 'Column|Position' FORMAT 9999
COLUMN INDEX_COLUMN HEADING 'Index|Column?' FORMAT A7

SELECT c.OWNER,
       c.COMPARISON_NAME,
       c.SCHEMA_NAME,
       c.OBJECT_NAME,
       o.COLUMN_NAME,
       o.COLUMN_POSITION,
       o.INDEX_COLUMN
  FROM DBA_COMPARISON c, DBA_COMPARISON_COLUMNS o
  WHERE c.OWNER           = o.OWNER AND
        c.COMPARISON_NAME = o.COMPARISON_NAME
  ORDER BY COMPARISON_NAME, COLUMN_POSITION;
```

Your output is similar to the following:

```
Comparison Comparison      Schema     Object     Column         Column Index
Owner      Name            Name       Name       Name         Position Column?
---------- --------------- ---------- ---------- ------------ -------- -------
ADMIN      COMPARE_CUSTOM  OE         ORDERS     ORDER_ID            1 Y
ADMIN      COMPARE_CUSTOM  OE         ORDERS     ORDER_DATE          2 N
ADMIN      COMPARE_CUSTOM  OE         ORDERS     ORDER_MODE          3 N
ADMIN      COMPARE_CUSTOM  OE         ORDERS     CUSTOMER_ID         4 N
ADMIN      COMPARE_CUSTOM  OE         ORDERS     ORDER_STATUS        5 N
ADMIN      COMPARE_CUSTOM  OE         ORDERS     ORDER_TOTAL         6 N
ADMIN      COMPARE_CUSTOM  OE         ORDERS     SALES_REP_ID        7 N
ADMIN      COMPARE_CUSTOM  OE         ORDERS     PROMOTION_ID        8 N
ADMIN      COMPARE_CYCLIC  OE         ORDERS     ORDER_ID            1 Y
ADMIN      COMPARE_CYCLIC  OE         ORDERS     ORDER_DATE          2 N
ADMIN      COMPARE_CYCLIC  OE         ORDERS     ORDER_MODE          3 N
ADMIN      COMPARE_CYCLIC  OE         ORDERS     CUSTOMER_ID         4 N
ADMIN      COMPARE_CYCLIC  OE         ORDERS     ORDER_STATUS        5 N
ADMIN      COMPARE_CYCLIC  OE         ORDERS     ORDER_TOTAL         6 N
ADMIN      COMPARE_CYCLIC  OE         ORDERS     SALES_REP_ID        7 N
ADMIN      COMPARE_CYCLIC  OE         ORDERS     PROMOTION_ID        8 N
.
.
.
```

See Also:

- "About Comparing and Converging Data" on page 13-1
- *Oracle Database PL/SQL Packages and Types Reference*

Viewing General Information About Each Scan in a Database

Each scan compares a bucket at the local database with a bucket at the remote database. The buckets being compared contain the same range of rows in the shared database object. The comparison results generated by a single execution of the COMPARE function can include multiple buckets and multiple scans. Each scan has a unique scan ID.

The query in this section shows the following information about each scan:

- The owner of the comparison that ran the scan
- The name of the comparison that ran the scan
- The schema that contains the database object compared by the scan
- The name of the database object compared by the scan
- The scan ID of the scan
- The status of the scan. The following status values are possible:
  - SUC indicates that the two buckets in the two tables matched the last time this data dictionary row was updated.
  - BUCKET DIF indicates that the two buckets in the two tables did not match. Each bucket consists of smaller buckets.
  - FINAL BUCKET DIF indicates that the two buckets in the two tables did not match. Neither bucket is composed of smaller buckets. Because the perform_row_dif parameter in the COMPARE function or the RECHECK function was set to FALSE, individual row differences were not identified for the bucket.
  - ROW DIF indicates that the two buckets in the two tables did not match. Neither bucket is composed of smaller buckets. Because the perform_row_dif parameter in the COMPARE function or the RECHECK function was set to TRUE, individual row differences were identified for the bucket.
- The number of rows compared in the scan
- The last time the scan was updated

To view this information, run the following query:

```
COLUMN OWNER HEADING 'Comparison|Owner' FORMAT A10
COLUMN COMPARISON_NAME HEADING 'Comparison|Name' FORMAT A15
COLUMN SCHEMA_NAME HEADING 'Schema|Name' FORMAT A6
COLUMN OBJECT_NAME HEADING 'Object|Name' FORMAT A6
COLUMN SCAN_ID HEADING 'Scan|ID' FORMAT 9999
COLUMN STATUS HEADING 'Scan|Status' FORMAT A10
COLUMN COUNT_ROWS HEADING 'Number|of|Rows' FORMAT 9999999
COLUMN SCAN_NULLS HEADING 'Scan|NULLs?' FORMAT A6
COLUMN LAST_UPDATE_TIME HEADING 'Last|Update' FORMAT A11

SELECT c.OWNER,
       c.COMPARISON_NAME,
       c.SCHEMA_NAME,
       c.OBJECT_NAME,
       s.SCAN_ID,
       s.STATUS,
       s.COUNT_ROWS,
       TO_CHAR(s.LAST_UPDATE_TIME, 'DD-MON-YYYY HH24:MI:SS') LAST_UPDATE_TIME
  FROM DBA_COMPARISON c, DBA_COMPARISON_SCAN s
```

```
  WHERE c.OWNER           = s.OWNER AND
        c.COMPARISON_NAME = s.COMPARISON_NAME
  ORDER BY SCAN_ID;
```

Your output is similar to the following:

```
                                                                Number
Comparison Comparison      Schema Object  Scan Scan             of Last
Owner      Name            Name   Name      ID Status         Rows Update
---------- --------------- ------ ------ ----- ---------- -------- -----------
ADMIN      COMPARE_SUBSET_ OE     ORDERS     1 BUCKET DIF          20-DEC-2006
           COLUMNS                                                   09:46:34
ADMIN      COMPARE_SUBSET_ OE     ORDERS     2 ROW DIF         105 20-DEC-2006
           COLUMNS                                                   09:46:34
ADMIN      COMPARE_SUBSET_ OE     ORDERS     3 ROW DIF           1 20-DEC-2006
           COLUMNS                                                   09:46:35
ADMIN      COMPARE_ORDERS  OE     ORDERS     4 BUCKET DIF          20-DEC-2006
                                                                     09:47:02
ADMIN      COMPARE_ORDERS  OE     ORDERS     5 FINAL BUCK      105 20-DEC-2006
                                                ET DIF               09:47:02
ADMIN      COMPARE_ORDERS  OE     ORDERS     6 FINAL BUCK        1 20-DEC-2006
                                                ET DIF               09:47:02
ADMIN      COMPARE_RANDOM  OE     ORDERS     7 SUC                 20-DEC-2006
                                                                     09:47:37
ADMIN      COMPARE_CYCLIC  OE     ORDERS     8 BUCKET DIF          20-DEC-2006
                                                                     09:48:22
ADMIN      COMPARE_CYCLIC  OE     ORDERS     9 ROW DIF         105 20-DEC-2006
                                                                     09:48:22
ADMIN      COMPARE_CUSTOM  OE     ORDERS    10 BUCKET DIF          20-DEC-2006
                                                                     09:49:15
ADMIN      COMPARE_CUSTOM  OE     ORDERS    11 ROW DIF          16 20-DEC-2006
                                                                     09:49:15
ADMIN      COMPARE_CUSTOM  OE     ORDERS    12 ROW DIF          13 20-DEC-2006
                                                                     09:49:15
```

When a scan has a status of `BUCKET DIF`, `FINAL BUCKET DIF`, or `ROW DIF`, you can converge the differences found in the scan by running the `CONVERGE` procedure and specifying the scan ID. However, to converge the all of the rows in the comparison results instead of the portion checked in a specific scan, specify the root scan ID for the comparison results when you run the `CONVERGE` procedure.

Also, when a scan shows that differences were found, you can recheck the scan using the `RECHECK` function. To recheck all of the rows in the comparison results, run the `RECHECK` function and specify the root scan ID for the comparison results.

See Also:

- "Viewing the Parent Scan ID and Root Scan ID for Each Scan in a Database" on page 13-26 for information about viewing the root scan for a scan
- "Converging a Shared Database Object" on page 13-30
- "Rechecking the Comparison Results for a Comparison" on page 13-34
- "About Comparing and Converging Data" on page 13-1 for more information about scans and buckets
- *Oracle Database PL/SQL Packages and Types Reference*

Viewing the Parent Scan ID and Root Scan ID for Each Scan in a Database

The query in this section shows the parent scan ID and root scan ID of each scan in the database. Specifically, the query shows the following information:

- The owner of the comparison that ran the scan
- The name of the comparison that ran the scan
- The schema that contains the database object compared by the scan
- The name of the database object compared by the scan
- The scan ID of the scan
- The scan ID of the scan's parent scan
- The scan ID of the scan's root scan

To view this information, run the following query:

```
COLUMN OWNER HEADING 'Comparison|Owner' FORMAT A10
COLUMN COMPARISON_NAME HEADING 'Comparison|Name' FORMAT A15
COLUMN SCHEMA_NAME HEADING 'Schema|Name' FORMAT A10
COLUMN OBJECT_NAME HEADING 'Object|Name' FORMAT A10
COLUMN SCAN_ID HEADING 'Scan|ID' FORMAT 9999
COLUMN PARENT_SCAN_ID HEADING 'Parent|Scan ID' FORMAT 9999
COLUMN ROOT_SCAN_ID HEADING 'Root|Scan ID' FORMAT 9999

SELECT c.OWNER,
       c.COMPARISON_NAME,
       c.SCHEMA_NAME,
       c.OBJECT_NAME,
       s.SCAN_ID,
       s.PARENT_SCAN_ID,
       s.ROOT_SCAN_ID
  FROM DBA_COMPARISON c, DBA_COMPARISON_SCAN s
  WHERE c.OWNER           = s.OWNER AND
        c.COMPARISON_NAME = s.COMPARISON_NAME
  ORDER BY s.SCAN_ID;
```

Your output is similar to the following:

```
Comparison Comparison      Schema     Object      Scan  Parent    Root
Owner      Name            Name       Name          ID Scan ID Scan ID
---------- --------------- ---------- ---------- ----- ------- -------
ADMIN      COMPARE_SUBSET_ OE         ORDERS         1               1
           COLUMNS
ADMIN      COMPARE_SUBSET_ OE         ORDERS         2       1       1
           COLUMNS
ADMIN      COMPARE_SUBSET_ OE         ORDERS         3       1       1
           COLUMNS
ADMIN      COMPARE_ORDERS  OE         ORDERS         4               4
ADMIN      COMPARE_ORDERS  OE         ORDERS         5       4       4
ADMIN      COMPARE_ORDERS  OE         ORDERS         6       4       4
ADMIN      COMPARE_RANDOM  OE         ORDERS         7               7
ADMIN      COMPARE_CYCLIC  OE         ORDERS         8               8
ADMIN      COMPARE_CYCLIC  OE         ORDERS         9       8       8
ADMIN      COMPARE_CUSTOM  OE         ORDERS        10              10
ADMIN      COMPARE_CUSTOM  OE         ORDERS        11      10      10
ADMIN      COMPARE_CUSTOM  OE         ORDERS        12      10      10
```

This output shows, for example, that the scan with scan ID `1` is the root scan in the comparison results for the `COMPARE_SUBSET_COLUMNS` comparison. Differences

were found in this root scan, and it was split into two smaller buckets. The scan with scan ID 2 and the scan with scan ID 3 are the scans for these smaller buckets.

To see if there were differences found in a specific scan, run the query in "Viewing General Information About Each Scan in a Database" on page 13-24. When you RECHECK for differences or CONVERGE differences in a shared database object, you specify the scan ID of the scan you want to recheck or converge. To recheck or converge all of the rows in the comparison results, specify the root scan ID for the comparison results.

See Also:

- "Converging a Shared Database Object" on page 13-30
- "Rechecking the Comparison Results for a Comparison" on page 13-34
- "About Comparing and Converging Data" on page 13-1
- *Oracle Database PL/SQL Packages and Types Reference*

Viewing Detailed Information About the Row Differences Found in a Scan

The queries in this section display detailed information about the row differences found in comparison results. To view the information in the queries in this section, the perform_row_dif parameter in the COMPARE function or the RECHECK function that performed the comparison must have been set to TRUE.

If this parameter was set to FALSE, then you can query the STATUS column in the DBA_COMPARISON_SCAN view to determine whether the scan found any differences, without showing detailed information about the differences. See "Viewing General Information About Each Scan in a Database" on page 13-24 for more information and a sample query.

The following query shows the total number of differences found for a scan with the scan ID of 8:

```
COLUMN OWNER HEADING 'Comparison Owner' FORMAT A16
COLUMN COMPARISON_NAME HEADING 'Comparison Name' FORMAT A25
COLUMN SCHEMA_NAME HEADING 'Schema Name' FORMAT A11
COLUMN OBJECT_NAME HEADING 'Object Name' FORMAT A11
COLUMN CURRENT_DIF_COUNT HEADING 'Differences' FORMAT 9999999

SELECT c.OWNER,
       c.COMPARISON_NAME,
       c.SCHEMA_NAME,
       c.OBJECT_NAME,
       s.CURRENT_DIF_COUNT
  FROM DBA_COMPARISON c, DBA_COMPARISON_SCAN s
  WHERE c.COMPARISON_NAME = s.COMPARISON_NAME AND
        c.OWNER           = s.OWNER AND
        s.SCAN_ID         = 8;
```

Your output is similar to the following:

```
Comparison Owner Comparison Name           Schema Name Object Name Differences
---------------- ------------------------- ----------- ----------- -----------
ADMIN            COMPARE_CYCLIC            OE          ORDERS                6
```

To view detailed information about each row difference found in the scan with scan ID 8 of the comparison results for the COMPARE_CYCLIC comparison, run the following query:

```
COLUMN COLUMN_NAME HEADING 'Index Column' FORMAT A15
COLUMN INDEX_VALUE HEADING 'Index Value' FORMAT A15
COLUMN LOCAL_ROWID HEADING 'Local Row Exists?' FORMAT A20
COLUMN REMOTE_ROWID HEADING 'Remote Row Exists?' FORMAT A20

SELECT c.COLUMN_NAME,
       r.INDEX_VALUE,
       DECODE(r.LOCAL_ROWID,
                NULL, 'No',
                      'Yes') LOCAL_ROWID,
       DECODE(r.REMOTE_ROWID,
                NULL, 'No',
                      'Yes') REMOTE_ROWID
  FROM DBA_COMPARISON_COLUMNS c,
       DBA_COMPARISON_ROW_DIF r,
       DBA_COMPARISON_SCAN s
  WHERE c.COMPARISON_NAME = 'COMPARE_CYCLIC' AND
        r.SCAN_ID         = s.SCAN_ID AND
        s.PARENT_SCAN_ID  = 8 AND
        r.STATUS          = 'DIF' AND
        c.INDEX_COLUMN    = 'Y' AND
        c.COMPARISON_NAME = r.COMPARISON_NAME AND
        c.OWNER           = r.OWNER
  ORDER BY r.INDEX_VALUE;
```

Your output is similar to the following:

```
Index Column    Index Value     Local Row Exists?    Remote Row Exists?
--------------- --------------- -------------------- --------------------
ORDER_ID        2366            Yes                  No
ORDER_ID        2385            Yes                  No
ORDER_ID        2396            Yes                  No
ORDER_ID        2425            Yes                  No
ORDER_ID        2440            Yes                  Yes
ORDER_ID        2450            Yes                  No
```

This output shows the index column for the table being compared and the index value for each row that is different in the shared database object. In this example, the index column is the primary key column for the `oe.orders` table (`order_id`). The output also shows the type of difference for each row:

- If `Local Row Exists?` and `Remote Row Exists?` are both `Yes` for a row, then the row exists in both instances of the database object, but the data in the row is different.
- If `Local Row Exists?` is `Yes` and `Remote Row Exists?` is `No` for a row, then the row exists in the local database object but not in the remote database object.
- If `Local Row Exists?` is `No` and `Remote Row Exists?` is `Yes` for a row, then the row exists in the remote database object but not in the local database object.

Viewing Information About the Rows Compared in Specific Scans

Each scan compares a range of rows in a shared database object. The query in this section provides the following information about the rows compared in each scan in the database:

- The owner of the comparison that ran the scan
- The name of the comparison that ran the scan
- The column position of the row values displayed by the query

- The minimum value for the range of rows compared by the scan
- The maximum value for the range of rows compared by the scan

A scan compares the row with the minimum value, the row with the maximum value, and all of the rows in between the minimum and maximum values in the database object. For each row returned by the query, the value displayed for the minimum value and the maximum value are the values for the column in the displayed the column position. The column position is an index column for the comparison.

To view this information, run the following query:

```
COLUMN OWNER HEADING 'Comparison|Owner' FORMAT A10
COLUMN COMPARISON_NAME HEADING 'Comparison|Name' FORMAT A22
COLUMN SCAN_ID HEADING 'Scan|ID' FORMAT 9999
COLUMN COLUMN_POSITION HEADING 'Column|Position' FORMAT 999
COLUMN MIN_VALUE HEADING 'Minimum|Value' FORMAT A15
COLUMN MAX_VALUE HEADING 'Maximum|Value' FORMAT A15

SELECT OWNER,
       COMPARISON_NAME,
       SCAN_ID,
       COLUMN_POSITION,
       MIN_VALUE,
       MAX_VALUE
  FROM DBA_COMPARISON_SCAN_VALUES
  ORDER BY SCAN_ID;
```

Your output is similar to the following:

```
Comparison Comparison              Scan   Column Minimum         Maximum
Owner      Name                      ID Position Value           Value
---------- ---------------------- ----- -------- --------------- ---------------
ADMIN      COMPARE_SUBSET_COLUMNS     1        1 2354            3000
ADMIN      COMPARE_SUBSET_COLUMNS     2        1 2354            2458
ADMIN      COMPARE_SUBSET_COLUMNS     3        1 3000            3000
ADMIN      COMPARE_ORDERS             4        1 2354            3000
ADMIN      COMPARE_ORDERS             5        1 2354            2458
ADMIN      COMPARE_ORDERS             6        1 3000            3000
ADMIN      COMPARE_RANDOM             7        1 2617.3400241505 2940.3400241505
                                                 667163579712423 667163579712423
                                                 44590999096     44590999096
ADMIN      COMPARE_CYCLIC             8        1 2354            2677
ADMIN      COMPARE_CYCLIC             9        1 2354            2458
ADMIN      COMPARE_CUSTOM            10        1 2430            2460
ADMIN      COMPARE_CUSTOM            11        1 2430            2445
ADMIN      COMPARE_CUSTOM            12        1 2446            2458
```

This output shows the rows that were compared in each scan. For some comparisons, the scan was split into smaller buckets, and the query shows the rows compared in each smaller bucket.

For example, consider the output for the comparison results of the `COMPARE_CUSTOM` comparison:

- Each scan in the comparison results displays column position `1`. To determine which column is in column position `1` for the scan, run the query in "Viewing the Columns Compared by Each Comparison in a Database" on page 13-22. In this example, the column in column position `1` for the `COMPARE_CUSTOM` comparison is the `order_id` column in the `oe.orders` table.

- Scan ID 10 is a root scan. This scan found differences, and the rows were split into two buckets that are represented by scan ID 11 and scan ID 12.
- Scan ID 11 compared the rows from the row with 2430 for order_id to the row with 2445 for order_id.
- Scan ID 12 compared the rows from the row with 2446 for order_id to the row with 2458 for order_id.

To recheck or converge the differences found in a scan, you can run the RECHECK function or CONVERGE procedure, respectively. Specify the scan ID of the scan you want to recheck or converge. To recheck or converge all of the rows in comparison results, specify the root scan ID for the comparison results.

See Also:

- "Converging a Shared Database Object" on page 13-30
- "Rechecking the Comparison Results for a Comparison" on page 13-34
- "About Comparing and Converging Data" on page 13-1
- *Oracle Database PL/SQL Packages and Types Reference*

Converging a Shared Database Object

The CONVERGE procedure in the DBMS_COMPARISON package synchronizes the portion of the database object compared by the specified comparison scan and returns information about the changes it made. The CONVERGE procedure only converges the differences identified in the specified scan. A scan might only identify differences in a subset of the rows or columns in a table, and differences might arise after the specified scan completed. In these cases, the CONVERGE procedure might not make the shared database object completely consistent.

To ensure that a scan has the most current differences, it is usually best to run the CONVERGE procedure as soon as possible after running the comparison scan that is being converged. Also, you should only converge rows that are not being updated on either database. For example, if the shared database object is updated by replication components, then only converge rows for which replication changes have already been applied and ensure that no new changes are in the process of being replicated for these rows.

Caution: If a scan identifies that a row is different in the shared database object at two databases, and the row is modified after the scan, then it can result in unexpected data in the row after the CONVERGE procedure is run.

This section contains the following examples:

- Converging a Shared Database Object for Consistency with the Local Object
- Converging a Shared Database Object for Consistency with the Remote Object
- Converging a Shared Database Object with a Session Tag Set

These examples converge the comparison results generated in "Comparing a Shared Database Object without Identifying Row Differences" on page 13-9. In that example, the comparison name is compare_orders and the returned scan ID is 4. If you

completed this example, then the scan ID returned on your system might have been different. Run the following query to determine the scan ID:

```
SELECT DISTINCT ROOT_SCAN_ID FROM DBA_COMPARISON_SCAN
  WHERE COMPARISON_NAME = 'COMPARE_ORDERS';
```

If multiple values are returned, then the comparison was run more than once. In this case, use the largest scan ID returned.

When you want to converge all of the rows in comparison results, specify the root scan ID for the comparison results. If, however, you want to converge a portion of the rows in comparison results, then you can specify the scan ID of the scan that contains differences you want to converge.

See Also:

- "Comparing a Shared Database Object at Two Databases" on page 13-7 for information about comparing database objects and comparison scans
- "Viewing General Information About Each Scan in a Database" on page 13-24 for a query that shows which scans found differences
- "Viewing the Parent Scan ID and Root Scan ID for Each Scan in a Database" on page 13-26 for a query that shows the root scan ID of each scan
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `CONVERGE` procedure

Converging a Shared Database Object for Consistency with the Local Object

The `converge_options` parameter in the `CONVERGE` procedure determines which database "wins" during a conversion. To specify that the local database wins, set the `converge_options` parameter to `DBMS_COMPARISON.CMP_CONVERGE_LOCAL_WINS`. When you specify that the local database wins, the data in the database object at the local database replaces the data in the database object at the remote database for each difference found in the specified comparison scan.

To converge a scan of the `compare_orders` comparison so that both database objects are consistent with the local database, complete the following steps:

1. In SQL*Plus, connect to the `comp1.example.com` database as the administrative user who owns the comparison. The user must also have access to the database link created in "Preparing To Compare and Converge a Shared Database Object" on page 13-6.

   See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.

2. Run the `CONVERGE` procedure:

```
SET SERVEROUTPUT ON
DECLARE
  scan_info    DBMS_COMPARISON.COMPARISON_TYPE;
BEGIN
  DBMS_COMPARISON.CONVERGE(
    comparison_name  => 'compare_orders',
    scan_id          => 4, -- Substitute the scan ID from your scan.
    scan_info        => scan_info,
    converge_options => DBMS_COMPARISON.CMP_CONVERGE_LOCAL_WINS);
DBMS_OUTPUT.PUT_LINE('Local Rows Merged: '||scan_info.loc_rows_merged);
```

```
DBMS_OUTPUT.PUT_LINE('Remote Rows Merged: '||scan_info.rmt_rows_merged);
DBMS_OUTPUT.PUT_LINE('Local Rows Deleted: '||scan_info.loc_rows_deleted);
DBMS_OUTPUT.PUT_LINE('Remote Rows Deleted: '||scan_info.rmt_rows_deleted);
END;
/
```

Your output is similar to the following:

```
Local Rows Merged: 0
Remote Rows Merged: 6
Local Rows Deleted: 0
Remote Rows Deleted: 1

PL/SQL procedure successfully completed.
```

Converging a Shared Database Object for Consistency with the Remote Object

The `converge_options` parameter in the `CONVERGE` procedure determines which database "wins" during a conversion. To specify that the remote database wins, set the `converge_options` parameter to `DBMS_COMPARISON.CMP_CONVERGE_REMOTE_WINS`. When you specify that the remote database wins, the data in the database object at the remote database replaces the data in the database object at the local database for each difference found in the specified comparison scan.

To converge a scan of the `compare_orders` comparison so that both database objects are consistent with the remote database, complete the following steps:

1. In SQL*Plus, connect to the `comp1.example.com` database as the administrative user who owns the comparison. The user must also have access to the database link created in "Preparing To Compare and Converge a Shared Database Object" on page 13-6.

   See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.

2. Run the `CONVERGE` procedure:

```
SET SERVEROUTPUT ON
DECLARE
  scan_info    DBMS_COMPARISON.COMPARISON_TYPE;
BEGIN
  DBMS_COMPARISON.CONVERGE(
    comparison_name  => 'compare_orders',
    scan_id          => 4, -- Substitute the scan ID from your scan.
    scan_info        => scan_info,
    converge_options => DBMS_COMPARISON.CMP_CONVERGE_REMOTE_WINS);
DBMS_OUTPUT.PUT_LINE('Local Rows Merged: '||scan_info.loc_rows_merged);
DBMS_OUTPUT.PUT_LINE('Remote Rows Merged: '||scan_info.rmt_rows_merged);
DBMS_OUTPUT.PUT_LINE('Local Rows Deleted: '||scan_info.loc_rows_deleted);
DBMS_OUTPUT.PUT_LINE('Remote Rows Deleted: '||scan_info.rmt_rows_deleted);
END;
/
```

   Your output is similar to the following:

```
Local Rows Merged: 2
Remote Rows Merged: 0
Local Rows Deleted: 5
Remote Rows Deleted: 0

PL/SQL procedure successfully completed.
```

Converging a Shared Database Object with a Session Tag Set

If the shared database object being converged is part of an Oracle Streams replication environment, then you can set a session tag so that changes made by the CONVERGE procedure are not replicated. Typically, changes made by the CONVERGE procedure should not be replicated to avoid change cycling, which means sending a change back to the database where it originated. In an Oracle Streams replication environment, you can use session tags to ensure that changes made by the CONVERGE procedure are not captured by Oracle Streams capture processes or synchronous captures and therefore not replicated.

To set a session tag in the session running the CONVERGE procedure, use the following procedure parameters:

- The local_converge_tag parameter sets a session tag at the local database. Set this parameter to a value that prevents replication when the remote database wins and the CONVERGE procedure makes changes to the local database.
- The remote_converge_tag parameter sets a session tag at the remote database. Set this parameter to a value that prevents replication when the local database wins and the CONVERGE procedure makes changes to the remote database.

The appropriate value for a session tag depends on the Oracle Streams replication environment. Set the tag to a value that prevents capture processes and synchronous captures from capturing changes made by the session.

> **See Also:** "Oracle Streams Tags in a Replication Environment" on page 10-6

The example in this section specifies that the local database wins the converge operation by setting the converge_options parameter to DBMS_COMPARISON.CMP_CONVERGE_LOCAL_WINS. Therefore, the example sets the remote_converge_tag parameter to the hexadecimal equivalent of '11'. The session tag can be set to any non-NULL value that prevents the changes made by the CONVERGE procedure to the remote database from being replicated.

To converge a scan of the compare_orders comparison so that the database objects are consistent with the local database and a session tag is set at the remote database, complete the following steps:

1. In SQL*Plus, connect to the comp1.example.com database as the administrative user who owns the comparison. The user must also have access to the database link created in "Preparing To Compare and Converge a Shared Database Object" on page 13-6.

   See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.

2. Run the CONVERGE procedure:

```
SET SERVEROUTPUT ON
DECLARE
  scan_info    DBMS_COMPARISON.COMPARISON_TYPE;
BEGIN
  DBMS_COMPARISON.CONVERGE(
    comparison_name     => 'compare_orders',
    scan_id             => 4, -- Substitute the scan ID from your scan.
    scan_info           => scan_info,
    converge_options    => DBMS_COMPARISON.CMP_CONVERGE_LOCAL_WINS,
```

```
    remote_converge_tag => HEXTORAW('11'));
  DBMS_OUTPUT.PUT_LINE('Local Rows Merged: '||scan_info.loc_rows_merged);
  DBMS_OUTPUT.PUT_LINE('Remote Rows Merged: '||scan_info.rmt_rows_merged);
  DBMS_OUTPUT.PUT_LINE('Local Rows Deleted: '||scan_info.loc_rows_deleted);
  DBMS_OUTPUT.PUT_LINE('Remote Rows Deleted: '||scan_info.rmt_rows_deleted);
  END;
  /
```

Your output is similar to the following:

```
Local Rows Merged: 0
Remote Rows Merged: 6
Local Rows Deleted: 0
Remote Rows Deleted: 1

PL/SQL procedure successfully completed.
```

Note: The `CREATE_COMPARISON` procedure also enables you to set local and remote convergence tag values. If a tag parameter in the `CONVERGE` procedure is non-`NULL`, then it takes precedence over the corresponding tag parameter in the `CREATE_COMPARISON` procedure. If a tag parameter in the `CONVERGE` procedure is `NULL`, then it is ignored, and the corresponding tag value in the `CREATE_COMPARISON` procedure is used.

Rechecking the Comparison Results for a Comparison

You can recheck a previous comparison scan by using the `RECHECK` function in the `DBMS_COMPARISON` package. The `RECHECK` function checks the current data in the database objects for differences that were recorded in the specified comparison scan.

For example, to recheck the results for scan ID `4` of a comparison named `compare_orders`, log in to SQL*Plus as the owner of the comparison, and run the following procedure:

```
SET SERVEROUTPUT ON
DECLARE
  consistent   BOOLEAN;
BEGIN
  consistent := DBMS_COMPARISON.RECHECK(
                comparison_name => 'compare_orders',
                scan_id         => 4);
  IF consistent=TRUE THEN
    DBMS_OUTPUT.PUT_LINE('No differences were found.');
  ELSE
    DBMS_OUTPUT.PUT_LINE('Differences were found.');
  END IF;
END;
/
```

Your output is similar to the following:

```
Differences were found.

PL/SQL procedure successfully completed.
```

The function returns TRUE if no differences were found or FALSE if differences were found. The compare_orders comparison is created in "Comparing a Shared Database Object without Identifying Row Differences" on page 13-9.

Note:

- The RECHECK function does not compare the shared database object for differences that were not recorded in the specified comparison scan. To check for those differences, run the COMPARE function.
- If the specified comparison scan did not complete successfully, then the RECHECK function starts where the comparison scan previously ended.

See Also: "Comparing a Shared Database Object at Two Databases" on page 13-7 for information about the compare function

Purging Comparison Results

You can purge the comparison results of one or more comparisons when they are no longer needed by using the PURGE_COMPARISON procedure in the DBMS_COMPARISON package. You can either purge all of the comparison results for a comparison or a subset of the comparison results. When comparison results are purged, they can no longer be used to recheck the comparison or converge divergent data. Also, information about the comparison results is removed from data dictionary views.

This section contains these topics:

- Purging All of the Comparison Results for a Comparison
- Purging the Comparison Results for a Specific Scan ID of a Comparison
- Purging the Comparison Results of a Comparison Before a Specified Time

See Also: "About Comparing and Converging Data" on page 13-1

Purging All of the Comparison Results for a Comparison

To purge all of the comparison results for a comparison, specify the comparison name in the comparison_name parameter, and specify the default value of NULL for the scan_id and purge_time parameters.

For example, to purge all of the comparison results for a comparison named compare_orders, log in to SQL*Plus as the owner of the comparison, and run the following procedure:

```
BEGIN
  DBMS_COMPARISON.PURGE_COMPARISON(
    comparison_name => 'compare_orders',
    scan_id         => NULL,
    purge_time      => NULL);
END;
/
```

Purging the Comparison Results for a Specific Scan ID of a Comparison

To purge the comparison results for a specific scan of a comparison, specify the comparison name in the `comparison_name` parameter, and specify the scan ID in the `scan_id` parameter. The specified scan ID must identify a root scan. The root scan in comparison results is the highest level parent scan. The root scan does not have a parent. You can identify root scan IDs by querying the `ROOT_SCAN_ID` column of the `DBA_COMPARISON_SCAN` data dictionary view.

When you run the `PURGE_COMPARISON` procedure and specify a root scan, the root scan is purged. In addition, all direct and indirect child scans of the specified root scan are purged. Results for other scans are not purged.

For example, to purge the comparison results for scan ID `4` of a comparison named `compare_orders`, log in to SQL*Plus as the owner of the comparison, and run the following procedure:

```
BEGIN
  DBMS_COMPARISON.PURGE_COMPARISON(
    comparison_name => 'compare_orders',
    scan_id         => 4); -- Substitute the scan ID from your scan.
END;
/
```

See Also:

- "Viewing the Parent Scan ID and Root Scan ID for Each Scan in a Database" on page 13-26
- *Oracle Database PL/SQL Packages and Types Reference*

Purging the Comparison Results of a Comparison Before a Specified Time

To purge the comparison results that were recorded on or before a specific date and time for a comparison, specify the comparison name in the `comparison_name` parameter, and specify the date and time in the `purge_time` parameter. Results are purged regardless of scan ID. Comparison results that were recorded after the specified date and time are retained.

For example, assume that the `NLS_TIMESTAMP_FORMAT` initialization parameter setting in the current session is `YYYY-MM-DD HH24:MI:SS`. To purge the results for any scans that were recorded before 1PM on August 16, 2006 for the `compare_orders` comparison, log in to SQL*Plus as the owner of the comparison, and run the following procedure:

```
BEGIN
  DBMS_COMPARISON.PURGE_COMPARISON(
    comparison_name => 'compare_orders',
    purge_time      => '2006-08-16 13:00:00');
END;
/
```

Dropping a Comparison

To drop a comparison and all of its comparison results, use the `DROP_COMPARISON` procedure in the `DBMS_COMPARISON` package. For example, to drop a comparison named `compare_subset_columns`, log in to SQL*Plus as the owner of the comparison, and run the following procedure:

```
exec DBMS_COMPARISON.DROP_COMPARISON('compare_subset_columns');
```

Using DBMS_COMPARISON in an Oracle Streams Replication Environment

This section describes the typical uses for the DBMS_COMPARISON package in an Oracle Streams replication environment. These uses are:

- Checking for Consistency After Instantiation
- Checking for Consistency in a Running Oracle Streams Replication Environment

Checking for Consistency After Instantiation

After an instantiation, you can use the DBMS_COMPARISON package to verify the consistency of the database objects that were instantiated. Typically, you should verify consistency before the Oracle Streams replication environment is replicating changes. Ensure that you check for consistency before you allow changes to the source database object and the instantiated database object. Changes to these database objects are identified as differences by the DBMS_COMPARISON package.

To verify the consistency of instantiated database objects, complete the following steps:

1. Create a comparison for each database object that was instantiated using the CREATE_COMPARISON procedure. Each comparison should specify the database object that was instantiated and its corresponding database object at the source database.

   When you run the CREATE_COMPARISON procedure, ensure that the comparison_mode, scan_mode, and scan_percent parameters are set to their default values of CMP_COMPARE_MODE_OBJECT, CMP_SCAN_MODE_FULL, and NULL, respectively.

2. Run the COMPARE function to compare each database object that was instantiated. The database objects are consistent if no differences are found.

   When you run the COMPARE function, ensure that the min_value, max_value, and perform_row_dif parameters are set to their default values of NULL, NULL, and FALSE, respectively.

3. If differences are found by the COMPARE function, then you can either re-instantiate the database objects or use the CONVERGE procedure to converge them. If you use the CONVERGE procedure, then typically the source database object should "win" during convergence.

4. When the comparison results show that the database objects are consistent, you can purge the comparison results using the PURGE_COMPARISON procedure.

See Also:

- "Comparing a Shared Database Object at Two Databases" on page 13-7 for instructions about creating a comparison with the CREATE_COMPARISON procedure and comparing database objects with the COMPARE function
- "Converging a Shared Database Object" on page 13-30
- "Purging Comparison Results" on page 13-35
- Chapter 8, "Instantiation and Oracle Streams Replication"

Checking for Consistency in a Running Oracle Streams Replication Environment

Oracle Streams replication environments continually replicate changes to database objects. Therefore, the following applies to the replicated database objects:

- Replicated database objects should be nearly synchronized most of the time because Oracle Streams components replicate and apply changes to keep them synchronized.
- If there are differences in replicated database objects, then Oracle Streams components will typically send and apply changes to synchronize the database objects in the near future. That is, a `COMPARE` function might show differences that are in the process of being replicated.

Because differences are expected in database objects while changes are being replicated, using the `DBMS_COMPARISON` package to compare replicated database objects can be challenging. For example, assume that there is an existing comparison that compares an entire table at two databases, and consider the following scenario:

1. A change is made to a row in the table at one of the databases.
2. The change is captured by an Oracle Streams capture process, but it has not yet been propagated to the other database.
3. The `COMPARE` function is run to compare the table tables at the two databases.
4. The `COMPARE` function identifies a difference in the row that was changed in Step 1.
5. The change is propagated and applied at the destination database. Therefore, the difference identified in Step 4 no longer exists.

When differences are found, and you suspect that the differences are transient, you can run the `RECHECK` function after some time has passed. If Oracle Streams has synchronized the database objects, then the differences will disappear.

If some rows in a replicated database object are constantly updated, then these rows might always show differences in comparison results. In this case, as you monitor the environment, ensure the following:

- No apply errors are accumulating at the destination database for these rows.
- The rows are being updated correctly by the Oracle Streams apply process at the destination database. You can query the table that contains the rows at the destination database to ensure that the replicated changes are being applied.

When both of these statements are true for the rows, then you can ignore differences in the comparison results for them.

Because the `COMPARE` function might show differences that are in the process of being replicated, it is best to run this function during times when there is the least amount of replication activity in your environment. During times of relatively little replication activity, comparison results show the following types of differences in an Oracle Streams replication environment:

- Differences resulting when rows are manually manipulated at only one database by an administrator or procedure. For example, an administrator or procedure might set a session tag before making changes, and the session tag might prevent a capture process from capturing the changes.
- Differences resulting from recovery situations in which data is lost at one database and must be identified and recovered from another database.

- Differences resulting from apply errors. In this case, the error transactions are not applied at one database because of apply errors.

In any of these situations, you can run the `CONVERGE` procedure to synchronize the database objects if it is appropriate. For example, if there are apply errors, and it is not easy to reexecute the error transactions, then you can use the `CONVERGE` procedure to synchronize the database objects.

See Also:

- "Comparing a Shared Database Object at Two Databases" on page 13-7 for instructions on creating a comparison with the `CREATE_COMPARISON` procedure and comparing database objects with the `COMPARE` function
- Rechecking the Comparison Results for a Comparison on page 13-34 for information about the `RECHECK` function
- "Converging a Shared Database Object" on page 13-30
- Chapter 10, "Oracle Streams Tags"
- *Oracle Streams Concepts and Administration* for information about apply errors

14

Managing Logical Change Records (LCRs)

This chapter contains instructions for managing logical change records (LCRs) in an Oracle Streams replication environment.

This chapter contains these topics:

- Requirements for Managing LCRs
- Constructing and Enqueuing LCRs
- Executing LCRs
- Managing LCRs Containing LOB Columns
- Managing LCRs Containing LONG or LONG RAW Columns

> **See Also:** *Oracle Database PL/SQL Packages and Types Reference* and *Oracle Streams Concepts and Administration* for more information about LCRs

Requirements for Managing LCRs

This section describes requirements for creating or modifying logical change records (LCRs). You can create an LCR using a constructor for an LCR type, and then enqueue the LCR into an persistent queue portion of an `ANYDATA` queue. Such an LCR is a persistent LCR.

Also, you can modify an LCR using an apply handler or a rule-based transformation. You can modify captured LCRs or persistent LCRs.

Ensure that you meet the following requirements when you manage an LCR:

- If you create or modify a row LCR, then ensure that 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 ensure that the `ddl_text` is consistent with the `base_table_name`, `base_table_owner`, `object_type`, `object_owner`, `object_name`, and `command_type` attributes.
- The following data types are allowed for columns in a user-constructed row LCR:
  - `CHAR`
  - `VARCHAR2`
  - `NCHAR`
  - `NVARCHAR2`
  - `NUMBER`

- DATE
- BINARY_FLOAT
- BINARY_DOUBLE
- RAW
- TIMESTAMP
- TIMESTAMP WITH TIME ZONE
- TIMESTAMP WITH LOCAL TIME ZONE
- INTERVAL YEAR TO MONTH
- INTERVAL DAY TO SECOND

These data types are the only data types allowed for columns in a user-constructed row LCR. However, you can use certain techniques to construct LCRs that contain LOB information. Also, LCRs captured by a capture process support more data types, while LCRs captured by a synchronous capture support fewer data types.

See Also:

- *Oracle Streams Concepts and Administration* for more information about apply handlers
- "Managing LCRs Containing LOB Columns" on page 14-11
- *Oracle Streams Concepts and Administration* for information about the data types captured by a capture process or a synchronous capture, and for information about rule-based transformations

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. Ensure that 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. Next, 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. This example assumes that you have configured an Oracle Streams administrator named `strmadmin` and granted this administrator `DBA` role.

Complete the following steps:

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

   See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.

2. Grant the Oracle Streams administrator `EXECUTE` privilege on the `DBMS_STREAMS_MESSAGING` package. For example:

```
GRANT EXECUTE ON DBMS_STREAMS_MESSAGING TO strmadmin;
```

Explicit EXECUTE privilege on the package is required because a procedure in the package is called within a PL/SQL procedure in Step 9. In this case, granting the privilege through a role is not sufficient.

3. In SQL*Plus, connect to the database as the Oracle Streams administrator.
4. Create an ANYDATA queue in an Oracle database.

```
BEGIN
  DBMS_STREAMS_ADM.SET_UP_QUEUE(
    queue_table          =>  'strm04_queue_table',
    storage_clause       =>  NULL,
    queue_name           =>  'strm04_queue');
END;
/
```

5. Create an apply process at the Oracle database to receive messages in the queue. Ensure that the apply_captured parameter is set to FALSE when you create the apply process, because the apply process will be applying persistent LCRs, not captured LCRs. Also, ensure that 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;
/
```

6. Create a positive rule set for the apply process and add a rule that applies DML changes to the hr.regions table made at the dbs1.example.com 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     =>  'dbs1.example.com',
    inclusion_rule      =>  TRUE);
END;
/
```

7. 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;
/
```

8. Start the apply process.

```
EXEC DBMS_APPLY_ADM.START_APPLY('strm04_apply');
```

9. Create a procedure called `construct_row_lcr` that constructs a row LCR and enqueues it into the queue created in Step 4.

```
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
  row_lcr        SYS.LCR$_ROW_RECORD;
BEGIN
  -- Construct the LCR based on information passed to procedure
  row_lcr := SYS.LCR$_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_STREAMS_MESSAGING.ENQUEUE(
    queue_name         =>  'strm04_queue',
    payload            =>  ANYDATA.ConvertObject(row_lcr));
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: *Oracle Database PL/SQL Packages and Types Reference* for more information about LCR constructors

10. Create and enqueue LCRs using the `construct_row_lcr` procedure created in Step 5.

    a. In SQL*Plus, connect to the database as the Oracle Streams administrator.

    b. Create a row LCR that inserts a row into the `hr.regions` table.

```
DECLARE
  newunit1  SYS.LCR$_ROW_UNIT;
  newunit2  SYS.LCR$_ROW_UNIT;
  newvals   SYS.LCR$_ROW_LIST;
BEGIN
  newunit1 := SYS.LCR$_ROW_UNIT(
    'region_id',
    ANYDATA.ConvertNumber(5),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  newunit2 := SYS.LCR$_ROW_UNIT(
    'region_name',
```

```
    ANYDATA.ConvertVarchar2('Moon'),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  newvals := SYS.LCR$_ROW_LIST(newunit1,newunit2);
construct_row_lcr(
  source_dbname  =>  'dbs1.example.com',
  cmd_type       =>  'INSERT',
  obj_owner      =>  'hr',
  obj_name       =>  'regions',
  old_vals       =>  NULL,
  new_vals       =>  newvals);
END;
/
COMMIT;
```

c. In SQL*Plus, connect to the database as the `hr` user.

d. Query the `hr.regions` table to view the applied row change. The row with a `region_id` of `5` should have `Moon` for the `region_name`.

```
SELECT * FROM hr.regions;
```

e. In SQL*Plus, connect to the database as the Oracle Streams administrator.

f. Create a row LCR that updates a row in the `hr.regions` table.

```
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',
    ANYDATA.ConvertNumber(5),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  oldunit2 := SYS.LCR$_ROW_UNIT(
    'region_name',
    ANYDATA.ConvertVarchar2('Moon'),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  oldvals := SYS.LCR$_ROW_LIST(oldunit1,oldunit2);
  newunit1 := SYS.LCR$_ROW_UNIT(
    'region_name',
    ANYDATA.ConvertVarchar2('Mars'),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  newvals := SYS.LCR$_ROW_LIST(newunit1);
construct_row_lcr(
  source_dbname  =>  'dbs1.example.com',
  cmd_type       =>  'UPDATE',
  obj_owner      =>  'hr',
  obj_name       =>  'regions',
  old_vals       =>  oldvals,
  new_vals       =>  newvals);
END;
```

```
/
COMMIT;
```

g. In SQL*Plus, connect to the database as the `hr` user.

h. Query the `hr.regions` table to view the applied row change. The row with a `region_id` of `5` should have `Mars` for the `region_name`.

```
SELECT * FROM hr.regions;
```

i. Create a row LCR that deletes a row from the `hr.regions` table.

```
DECLARE
  oldunit1  SYS.LCR$_ROW_UNIT;
  oldunit2  SYS.LCR$_ROW_UNIT;
  oldvals   SYS.LCR$_ROW_LIST;
BEGIN
  oldunit1 := SYS.LCR$_ROW_UNIT(
    'region_id',
    ANYDATA.ConvertNumber(5),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  oldunit2 := SYS.LCR$_ROW_UNIT(
    'region_name',
    ANYDATA.ConvertVarchar2('Mars'),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  oldvals := SYS.LCR$_ROW_LIST(oldunit1,oldunit2);
construct_row_lcr(
  source_dbname  =>  'dbs1.example.com',
  cmd_type       =>  'DELETE',
  obj_owner      =>  'hr',
  obj_name       =>  'regions',
  old_vals       =>  oldvals,
  new_vals       =>  NULL);
END;
/
COMMIT;
```

j. In SQL*Plus, connect to the database as the `hr` user.

k. Query the `hr.regions` table to view the applied row change. The row with a `region_id` of `5` should have been deleted.

```
SELECT * FROM hr.regions;
```

Executing LCRs

There are separate `EXECUTE` member procedures for row LCRs and DDL LCRs. These member procedures execute an LCR under the security domain of the current user. When an LCR is executed successfully, the change recorded in the LCR is made to the local database. The following sections describe executing row LCRs and DDL LCRs:

- Executing Row LCRs
- Executing DDL LCRs

Executing Row LCRs

The EXECUTE member procedure for row LCRs is a subprogram of the LCR$_ROW_RECORD type. When the EXECUTE member procedure is run on a row LCR, the row LCR is executed. If the row LCR is executed by an apply process, then any apply process handlers that would be run for the LCR are not run.

The EXECUTE member procedure can be run on a row LCR under any of the following conditions:

- The LCR is being processed by an apply handler.
- The LCR is in a queue and was last enqueued by an apply process, an application, or a user.
- The LCR has been constructed using the LCR$_ROW_RECORD constructor function but has not been enqueued.
- The LCR is in the error queue.

When you run the EXECUTE member procedure on a row LCR, the conflict_resolution parameter controls whether conflict resolution is performed. Specifically, if the conflict_resolution parameter is set to TRUE, then any conflict resolution defined for the table being changed is used to resolve conflicts resulting from the execution of the LCR. If the conflict_resolution parameter is set to FALSE, then conflict resolution is not used. If the conflict_resolution parameter is not set or is set to NULL, then an error is raised.

Note: A custom rule-based transformation should not run the EXECUTE member procedure on a row LCR. Doing so could execute the row LCR outside of its transactional context.

See Also:

- *Oracle Streams Concepts and Administration*
- *Oracle Database PL/SQL Packages and Types Reference* for more information about row LCRs and the LCR$_ROW_RECORD type

Example of Constructing and Executing Row LCRs

The example in this section creates PL/SQL procedures to insert, update, and delete rows in the hr.jobs table by constructing and executing row LCRs. The row LCRs are executed without being enqueued or processed by an apply process. This example assumes that you have configured an Oracle Streams administrator named strmadmin and granted this administrator DBA role.

Complete the following steps:

1. In SQL*Plus, connect to the database as the Oracle Streams administrator.

   See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.

2. Create a PL/SQL procedure named execute_row_lcr that executes a row LCR:

```
CREATE OR REPLACE PROCEDURE execute_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
  xrow_lcr  SYS.LCR$_ROW_RECORD;
BEGIN
  -- Construct the row LCR based on information passed to procedure
  xrow_lcr := SYS.LCR$_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);
  -- Execute the row LCR
  xrow_lcr.EXECUTE(FALSE);
END execute_row_lcr;
/
```

3. Create a PL/SQL procedure named `insert_job_lcr` that executes a row LCR that inserts a row into the `hr.jobs` table:

```
CREATE OR REPLACE PROCEDURE insert_job_lcr(
                j_id     VARCHAR2,
                j_title  VARCHAR2,
                min_sal  NUMBER,
                max_sal  NUMBER) AS
  xrow_lcr   SYS.LCR$_ROW_RECORD;
  col1_unit  SYS.LCR$_ROW_UNIT;
  col2_unit  SYS.LCR$_ROW_UNIT;
  col3_unit  SYS.LCR$_ROW_UNIT;
  col4_unit  SYS.LCR$_ROW_UNIT;
  newvals    SYS.LCR$_ROW_LIST;
BEGIN
  col1_unit := SYS.LCR$_ROW_UNIT(
    'job_id',
    ANYDATA.ConvertVarchar2(j_id),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  col2_unit := SYS.LCR$_ROW_UNIT(
    'job_title',
    ANYDATA.ConvertVarchar2(j_title),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  col3_unit := SYS.LCR$_ROW_UNIT(
    'min_salary',
    ANYDATA.ConvertNumber(min_sal),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  col4_unit := SYS.LCR$_ROW_UNIT(
    'max_salary',
    ANYDATA.ConvertNumber(max_sal),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  newvals := SYS.LCR$_ROW_LIST(col1_unit,col2_unit,col3_unit,col4_unit);
  -- Execute the row LCR
  execute_row_lcr(
    source_dbname => 'DB1.EXAMPLE.COM',
    cmd_type      => 'INSERT',
```

```
    obj_owner     => 'HR',
    obj_name      => 'JOBS',
    old_vals      => NULL,
    new_vals      => newvals);
END insert_job_lcr;
/
```

4. Create a PL/SQL procedure named `update_max_salary_lcr` that executes a row LCR that updates the `max_salary` value for a row in the `hr.jobs` table:

```
CREATE OR REPLACE PROCEDURE update_max_salary_lcr(
                 j_id         VARCHAR2,
                 old_max_sal NUMBER,
                 new_max_sal NUMBER) AS
  xrow_lcr      SYS.LCR$_ROW_RECORD;
  oldcol1_unit  SYS.LCR$_ROW_UNIT;
  oldcol2_unit  SYS.LCR$_ROW_UNIT;
  newcol1_unit  SYS.LCR$_ROW_UNIT;
  oldvals       SYS.LCR$_ROW_LIST;
  newvals       SYS.LCR$_ROW_LIST;
BEGIN
  oldcol1_unit := SYS.LCR$_ROW_UNIT(
    'job_id',
    ANYDATA.ConvertVarchar2(j_id),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  oldcol2_unit := SYS.LCR$_ROW_UNIT(
    'max_salary',
    ANYDATA.ConvertNumber(old_max_sal),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  oldvals := SYS.LCR$_ROW_LIST(oldcol1_unit,oldcol2_unit);
  newcol1_unit := SYS.LCR$_ROW_UNIT(
    'max_salary',
    ANYDATA.ConvertNumber(new_max_sal),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  newvals := SYS.LCR$_ROW_LIST(newcol1_unit);
  -- Execute the row LCR
  execute_row_lcr(
    source_dbname => 'DB1.EXAMPLE.COM',
    cmd_type      => 'UPDATE',
    obj_owner     => 'HR',
    obj_name      => 'JOBS',
    old_vals      => oldvals,
    new_vals      => newvals);
END update_max_salary_lcr;
/
```

5. Create a PL/SQL procedure named `delete_job_lcr` that executes a row LCR that deletes a row from the `hr.jobs` table:

```
CREATE OR REPLACE PROCEDURE delete_job_lcr(j_id VARCHAR2) AS
  xrow_lcr   SYS.LCR$_ROW_RECORD;
  col1_unit  SYS.LCR$_ROW_UNIT;
  oldvals    SYS.LCR$_ROW_LIST;
```

```
BEGIN
  col1_unit := SYS.LCR$_ROW_UNIT(
    'job_id',
    ANYDATA.ConvertVarchar2(j_id),
    DBMS_LCR.NOT_A_LOB,
    NULL,
    NULL);
  oldvals := SYS.LCR$_ROW_LIST(col1_unit);
  -- Execute the row LCR
  execute_row_lcr(
    source_dbname => 'DB1.EXAMPLE.COM',
    cmd_type      => 'DELETE',
    obj_owner     => 'HR',
    obj_name      => 'JOBS',
    old_vals      => oldvals,
    new_vals      => NULL);
END delete_job_lcr;
/
```

6. Insert a row into the `hr.jobs` table using the `insert_job_lcr` procedure:

```
EXEC insert_job_lcr('BN_CNTR','BEAN COUNTER',5000,10000);
```

7. Select the inserted row in the `hr.jobs` table:

```
SELECT * FROM hr.jobs WHERE job_id = 'BN_CNTR';

JOB_ID     JOB_TITLE                          MIN_SALARY MAX_SALARY
---------- ---------------------------------- ---------- ----------
BN_CNTR    BEAN COUNTER                             5000      10000
```

8. Update the `max_salary` value for the row inserted into the `hr.jobs` table in Step 6 using the `update_max_salary_lcr` procedure:

```
EXEC update_max_salary_lcr('BN_CNTR',10000,12000);
```

9. Select the updated row in the `hr.jobs` table:

```
SELECT * FROM hr.jobs WHERE job_id = 'BN_CNTR';

JOB_ID     JOB_TITLE                          MIN_SALARY MAX_SALARY
---------- ---------------------------------- ---------- ----------
BN_CNTR    BEAN COUNTER                             5000      12000
```

10. Delete the row inserted into the `hr.jobs` table in Step 6 using the `delete_job_lcr` procedure:

```
EXEC delete_job_lcr('BN_CNTR');
```

11. Select the deleted row in the `hr.jobs` table:

```
SELECT * FROM hr.jobs WHERE job_id = 'BN_CNTR';

no rows selected
```

Executing DDL LCRs

The `EXECUTE` member procedure for DDL LCRs is a subprogram of the `LCR$_DDL_RECORD` type. When the `EXECUTE` member procedure is run on a DDL LCR, the LCR is executed, and any apply process handlers that would be run for the LCR are not

run. The EXECUTE member procedure for DDL LCRs can be invoked only in an apply handler for an apply process.

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:

- *Oracle Streams Concepts and Administration*
- *Oracle Database PL/SQL Packages and Types Reference* for more information about DDL LCRs and the LCR$_DDL_RECORD type

Managing LCRs Containing LOB Columns

LOB data types can be present in row LCRs captured by a capture process, but these data types are represented by other data types. LOB data types cannot be present in row LCRs captured by synchronous captures. Certain LOB data types cannot be present in row LCRs constructed by users. Table 14–1 shows the LCR representation for these data types and whether these data types can be present in row LCRs.

Table 14–1 LOB Data Type Representations in Row LCRs

Data Type	Row LCR Representation	Can Be Present in a Row LCR Captured by a Capture Process?	Can Be Present in a Row LCR Captured by a Synchronous Capture?	Can Be Present in a Row LCR Constructed by a User?
Fixed-width CLOB	VARCHAR2	Yes	No	Yes
Variable-width CLOB	RAW in AL16UTF16 character set	Yes	No	No
NCLOB	RAW in AL16UTF16 character set	Yes	No	No
BLOB	RAW	Yes	No	Yes
XMLType stored as CLOB	RAW	Yes	No	No

The following are general considerations for row changes involving LOB data types in an Oracle Streams environment:

- A row change involving a LOB column can be captured, propagated, and applied as several row LCRs.
- Rules used to evaluate these row LCRs must be deterministic, so that either all of the row LCRs corresponding to the row change cause a rule in a rule set to evaluate to TRUE, or none of them do.

The following sections contain information about the requirements you must meet when constructing or processing LOB columns, about apply process behavior for LCRs containing LOB columns, and about LOB assembly. There is also an example that constructs and enqueues LCRs containing LOB columns.

This section contains the following topics:

- Apply Process Behavior for Direct Apply of LCRs Containing LOBs
- LOB Assembly and Custom Apply of LCRs Containing LOB Columns
- Requirements for Constructing and Processing LCRs Containing LOB Columns

See Also:

- *Oracle Database SecureFiles and Large Objects Developer's Guide* for more information about LOBs
- *Oracle Streams Extended Examples* for an example that constructs and enqueues LCRs that contain LOBs

Apply Process Behavior for Direct Apply of LCRs Containing LOBs

An apply process behaves in the following ways when it applies an LCR that contains a LOB column directly (without the use of an apply handler):

- 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.

LOB Assembly and Custom Apply of LCRs Containing LOB Columns

A change to a row in a table that does not include any LOB columns results in a single row LCR, but a change to a row that includes one or more LOB columns can result in multiple row LCRs. An apply process that does not send row LCRs that contain LOB columns to an apply handler can apply these row LCRs directly. However, before Oracle Database 10*g* Release 2, custom processing of row LCRs that contain LOB columns was complicated because apply handlers had to be configured to process multiple LCRs correctly for a single row change.

In Oracle Database 10*g* Release 2 and later, **LOB assembly** simplifies custom processing of row LCRs with LOB columns that were captured by a capture process. LOB assembly automatically combines multiple captured row LCRs resulting from a change to a row with LOB columns into one row LCR. An apply process passes this single row LCR to a DML handler or error handler when LOB assembly is enabled.

Also, after LOB assembly, the LOB column values are represented by LOB locators, not by VARCHAR2 or RAW data type values. To enable LOB assembly for a procedure DML or error handler, set the assemble_lobs parameter to TRUE in the DBMS_APPLY_ADM.SET_DML_HANDLER procedure. LOB assembly is always enabled for statement DML handlers.

If the assemble_lobs parameter is set to FALSE for a DML or error handler, then LOB assembly is disabled and multiple row LCRs are passed to the handler for a change to a single row with LOB columns. Table 14–2 shows Oracle Streams behavior when LOB assembly is disabled. Specifically, the table shows the LCRs passed to a procedure DML handler or error handler resulting from a change to a single row with LOB columns.

Table 14–2 Oracle Streams Behavior with LOB Assembly Disabled

Original Row Change	First Set of LCRs	Second Set of LCRs	Third Set of LCRs	Final LCR
INSERT	One INSERT LCR	One or more LOB WRITE LCRs	One or more LOB TRIM LCRs	UPATE
UPDATE	One UPDATE LCR	One or more LOB WRITE LCRs	One or more LOB TRIM LCRs	UPATE
DELETE	One DELETE LCR	N/A	N/A	N/A
DBMS_LOB.WRITE	One or more LOB WRITE LCRs	N/A	N/A	N/A
DBMS_LOB.TRIM	One LOB TRIM LCR	N/A	N/A	N/A
DBMS_LOB.ERASE	One LOB ERASE LCR	N/A	N/A	N/A

Table 14–3 shows Oracle Streams behavior when LOB assembly is enabled. Specifically, the table shows the row LCR passed to a DML handler or error handler resulting from a change to a single row with LOB columns.

Table 14–3 Oracle Streams Behavior with LOB Assembly Enabled

Original Row Change	Single LCR
INSERT	INSERT
UPDATE	UPDATE
DELETE	DELETE
DBMS_LOB.WRITE	LOB WRITE
DBMS_LOB.TRIM	LOB TRIM
DBMS_LOB.ERASE	LOB ERASE

When LOB assembly is enabled, a DML or error handler can modify LOB columns in a row LCR. Within the PL/SQL procedure specified as a DML or error handler, the preferred way to perform operations on a LOB is to use a subprogram in the DBMS_LOB package. If a row LCR contains a LOB column that is NULL, then a new LOB locator must replace the NULL. If a row LCR will be applied with the EXECUTE member procedure, then use the ADD_COLUMN, SET_VALUE, and SET_VALUES member procedures for row LCRs to make changes to a LOB.

When LOB assembly is enabled, LOB assembly converts non-NULL LOB columns in persistent LCRs into LOB locators. However, LOB assembly does not combine

multiple persistent row LCRs into a single row LCR. For example, for persistent row LCRs, LOB assembly does not combine multiple `LOB WRITE` row LCRs following an `INSERT` row LCR into a single `INSERT` row LCR.

See Also:

- *Oracle Streams Concepts and Administration* for more information about apply handlers
- *Oracle Database SecureFiles and Large Objects Developer's Guide* and *Oracle Database PL/SQL Packages and Types Reference* for more information about using the `DBMS_LOB` package
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `ADD_COLUMN`, `SET_VALUE`, and `SET_VALUES` member procedures for row LCRs

LOB Assembly Considerations

The following are issues to consider when you use LOB assembly:

- To use a DML or error handler to process assembled LOBs at multiple destination databases, LOB assembly must assemble the LOBs separately on each destination database.
- Row LCRs captured on a database running a release of Oracle before Oracle Database 10*g* Release 2 cannot be assembled by LOB assembly.
- Row LCRs captured on a database running Oracle Database 10*g* Release 2 or later with a compatibility level lower than `10.2.0` cannot be assembled by LOB assembly.
- The compatibility level of the database running an apply handler must be `10.2.0` or higher to specify LOB assembly for the apply handler.
- Row LCRs from a table containing any `LONG` or `LONG RAW` columns cannot be assembled by LOB assembly.
- The `SET_ENQUEUE_DESTINATION` and the `SET_EXECUTE` procedures in the `DBMS_APPLY_ADM` package always operate on original, nonassembled row LCRs. Therefore, for row LCRs that contain LOB columns, the original, nonassembled row LCRs are enqueued or executed, even if these row LCRs are assembled separately for an apply handler at the destination database.
- If rule-based transformations were performed on row LCRs that contain LOB columns during capture, propagation, or apply, then an apply handler operates on the transformed row LCRs. If there are `LONG` or `LONG RAW` columns at a source database, and a rule-based transformation uses the `CONVERT_LONG_TO_LOB_CHUNK` member function for row LCRs to convert them to LOBs, then LOB assembly can be enabled for apply handlers that operate on these row LCRs.
- When a row LCR contains one or more `XMLType` columns, any `XMLType` and LOB columns in the row LCR are always assembled, even if the `assemble_lobs` parameter is set to `FALSE` for a DML or error handler.

See Also:

- *Oracle Database Reference* and *Oracle Database Upgrade Guide* for more information database compatibility
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the subprograms in the `DBMS_APPLY_ADM` package

LOB Assembly Example

This section contains an example that uses LOB assembly with a procedure DML handler. The example scenario involves a company that shares the `oe.production_information` table at several databases, but only some of these databases are used for the company's online World Wide Web catalog. The company wants to store a photograph of each product in the catalog databases, but, to save space, it does not want to store these photographs at the non catalog databases.

To accomplish this goal, a procedure DML handler at a catalog destination database can add a column named `photo` of data type `BLOB` to each `INSERT` and `UPDATE` made to the `product_information` table at a source database. The source database does not include the `photo` column in the table. The procedure DML handler is configured to use an existing photograph at the destination for updates and inserts.

The company also wants to add a `product_long_desc` to the `oe.product_information` table at all databases. This table already has a `product_description` column that contains short descriptions. The `product_long_desc` column is of `CLOB` data type and contains detailed descriptions. The detailed descriptions are in English, but one of the company databases is used to display the company catalog in Spanish. Therefore, the procedure DML handler updates the `product_long_desc` column so that the long description is in the correct language.

The following steps configure a procedure DML handler that uses LOB assembly to accomplish the goals described previously:

Step 1 Add the photo Column to the product_information Table

The following statement adds the `photo` column to the `product_information` table at the destination database:

```
ALTER TABLE oe.product_information ADD(photo BLOB);
```

Step 2 Add the product_long_desc Column to the product_information Table

The following statement adds the `product_long_desc` column to the `product_information` table at all of the databases in the environment:

```
ALTER TABLE oe.product_information ADD(product_long_desc CLOB);
```

Step 3 Create the PL/SQL Procedure for the Procedure DML Handler

This example creates the `convert_product_information` procedure. This procedure will be used for the procedure DML handler. This procedure assumes that the following user-created PL/SQL subprograms exist:

- The `get_photo` procedure obtains a photo in `BLOB` format from a URL or table based on the `product_id` and updates the `BLOB` locator that has been passed in as an argument.
- The `get_product_long_desc` procedure has an `IN` argument of `product_id` and an `IN OUT` argument of `product_long_desc` and translates the `product_`

long_desc into Spanish or obtains the Spanish replacement description and updates product_long_desc.

The following code creates the convert_product_information procedure:

```
CREATE OR REPLACE PROCEDURE convert_product_information(in_any IN ANYDATA)
IS
  lcr                      SYS.LCR$_ROW_RECORD;
  rc                       PLS_INTEGER;
  product_id_anydata       ANYDATA;
  photo_anydata            ANYDATA;
  long_desc_anydata        ANYDATA;
  tmp_photo                BLOB;
  tmp_product_id           NUMBER;
  tmp_prod_long_desc       CLOB;
  tmp_prod_long_desc_src   CLOB;
  tmp_prod_long_desc_dest  CLOB;
  t                        PLS_INTEGER;
BEGIN
  -- Access LCR
  rc := in_any.GETOBJECT(lcr);
  product_id_anydata := lcr.GET_VALUE('OLD', 'PRODUCT_ID');
  t := product_id_anydata.GETNUMBER(tmp_product_id);
  IF ((lcr.GET_COMMAND_TYPE = 'INSERT') or (lcr.GET_COMMAND_TYPE = 'UPDATE')) THEN
    -- If there is no photo column in the lcr then it must be added
    photo_anydata := lcr.GET_VALUE('NEW', 'PHOTO');
    -- Check if photo has been sent and if so whether it is NULL
    IF (photo_anydata is NULL) THEN
      tmp_photo := NULL;
      ELSE
      t := photo_anydata.GETBLOB(tmp_photo);
    END IF;
    -- If tmp_photo is NULL then a new temporary LOB must be created and
    -- updated with the photo if it exists
    IF (tmp_photo is NULL) THEN
      DBMS_LOB.CREATETEMPORARY(tmp_photo, TRUE);
      get_photo(tmp_product_id, tmp_photo);
    END IF;
    -- If photo column did not exist then it must be added
    IF (photo_anydata is NULL) THEN
      lcr.ADD_COLUMN('NEW', 'PHOTO', ANYDATA.CONVERTBLOB(tmp_photo));
      -- Else the existing photo column must be set to the new photo
      ELSE
        lcr.SET_VALUE('NEW', 'PHOTO', ANYDATA.CONVERTBLOB(tmp_photo));
    END IF;
    long_desc_anydata := lcr.GET_VALUE('NEW', 'PRODUCT_LONG_DESC');
    IF (long_desc_anydata is NULL) THEN
      tmp_prod_long_desc_src := NULL;
      ELSE
      t := long_desc_anydata.GETCLOB(tmp_prod_long_desc_src);
    END IF;
    IF (tmp_prod_long_desc_src IS NOT NULL) THEN
      get_product_long_desc(tmp_product_id, tmp_prod_long_desc);
    END IF;
    -- If tmp_prod_long_desc IS NOT NULL, then use it to update the LCR
    IF (tmp_prod_long_desc IS NOT NULL) THEN
      lcr.SET_VALUE('NEW', 'PRODUCT_LONG_DESC',
                    ANYDATA.CONVERTCLOB(tmp_prod_long_desc_dest));
    END IF;
  END IF;
```

```
  -- DBMS_LOB operations also are executed
  -- Inserts and updates invoke all changes
  lcr.EXECUTE(TRUE);
END;
/
```

Step 4 Set the Procedure DML Handler for the Apply Process

This step sets the `convert_product_information` procedure as the procedure DML handler at the destination database for `INSERT`, `UPDATE`, and `LOB_UPDATE` operations. Notice that the `assemble_lobs` parameter is set to `TRUE` each time the `SET_DML_HANDLER` procedure is run.

```
BEGIN
  DBMS_APPLY_ADM.SET_DML_HANDLER(
    object_name         => 'oe.product_information',
    object_type         => 'TABLE',
    operation_name      => 'INSERT',
    error_handler       => FALSE,
    user_procedure      => 'strmadmin.convert_product_information',
    apply_database_link => NULL,
    assemble_lobs       => TRUE);
  DBMS_APPLY_ADM.SET_DML_HANDLER(
    object_name         => 'oe.product_information',
    object_type         => 'TABLE',
    operation_name      => 'UPDATE',
    error_handler       => FALSE,
    user_procedure      => 'strmadmin.convert_product_information',
    apply_database_link => NULL,
    assemble_lobs       => TRUE);
  DBMS_APPLY_ADM.SET_DML_HANDLER(
    object_name         => 'oe.product_information',
    object_type         => 'TABLE',
    operation_name      => 'LOB_UPDATE',
    error_handler       => FALSE,
    user_procedure      => 'strmadmin.convert_product_information',
    apply_database_link => NULL,
    assemble_lobs       => TRUE);
END;
/
```

Step 5 Query the DBA_APPLY_DML_HANDLERS View

To ensure that the procedure DML handler is set properly for the `oe.product_information` table, run the following query:

```
COLUMN OBJECT_OWNER HEADING 'Table|Owner' FORMAT A5
COLUMN OBJECT_NAME HEADING 'Table Name' FORMAT A20
COLUMN OPERATION_NAME HEADING 'Operation' FORMAT A10
COLUMN USER_PROCEDURE HEADING 'Handler Procedure' FORMAT A25
COLUMN ASSEMBLE_LOBS HEADING 'LOB Assembly?' FORMAT A15

SELECT OBJECT_OWNER,
       OBJECT_NAME,
       OPERATION_NAME,
       USER_PROCEDURE,
       ASSEMBLE_LOBS
  FROM DBA_APPLY_DML_HANDLERS;
```

Your output looks similar to the following:

```
Table
Owner Table Name           Operation  Handler Procedure         LOB Assembly?
----- -------------------- ---------- ------------------------- ---------------
OE    PRODUCT_INFORMATION  INSERT     "STRMADMIN"."CONVERT_PROD Y
                                      UCT_INFORMATION"

OE    PRODUCT_INFORMATION  UPDATE     "STRMADMIN"."CONVERT_PROD Y
                                      UCT_INFORMATION"

OE    PRODUCT_INFORMATION  LOB_UPDATE "STRMADMIN"."CONVERT_PROD Y
                                      UCT_INFORMATION"
```

Notice that the correct procedure, `convert_product_information`, is used for each operation on the table. Also, notice that each handler uses LOB assembly.

Requirements for Constructing and Processing LCRs Containing LOB Columns

If your environment produces row LCRs that contain LOB columns, then you must meet the requirements in the following sections when you construct or process these LCRs:

- Requirements for Constructing and Processing LCRs Without LOB Assembly
- Requirements for Apply Handler Processing of LCRs with LOB Assembly
- Requirements for Rule-Based Transformation Processing of LCRs with LOBs

> **See Also:** *Oracle Streams Extended Examples* for an example that constructs and enqueues LCRs that contain LOBs

Requirements for Constructing and Processing LCRs Without LOB Assembly

The following requirements must be met when you are constructing LCRs with LOB columns and when you are processing LOB columns with a DML or error handler that has LOB assembly disabled:

- Do not modify LOB column data in a row LCR with a procedure DML handler or error handler that has LOB assembly disabled. However, you can modify non-LOB columns in row LCRs with a DML or error handler.
- Do not allow LCRs from a table that contains LOB columns to be processed by an apply handler that is invoked only for specific operations. For example, an apply handler that is invoked only for `INSERT` operations should not process LCRs from a table with one or more LOB columns.
- 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`.
- A LOB column in a user-constructed row LCR must be either a `BLOB` or a fixed-width `CLOB`. You cannot construct a row LCR with the following types of LOB columns: `NCLOB` or variable-width `CLOB`.
- `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 an `ANYDATA`.

An apply process performs all validation of these requirements. If these requirements are not met, then a row LCR containing LOB columns cannot be applied by an apply process nor processed by an apply handler. In this case, the LCR is moved to the error queue with the rest of the LCRs in the same transaction.

See Also:

- "Constructing and Enqueuing LCRs" on page 14-2
- *Oracle Streams Concepts and Administration* for more information about apply handlers

Requirements for Apply Handler Processing of LCRs with LOB Assembly

The following requirements must be met when you are processing LOB columns with a DML or error handler that has LOB assembly enabled:

- Do not use the following row LCR member procedures on LOB columns in row LCRs that contain assembled LOBs:
  - `SET_LOB_INFORMATION`
  - `SET_LOB_OFFSET`
  - `SET_LOB_OPERATION_SIZE`

  An error is raised if one of these procedures is used on a LOB column in a row LCR.
- Row LCRs constructed by LOB assembly cannot be enqueued by a procedure DML handler or error handler. However, even when LOB assembly is enabled for one or more handlers at a destination database, the original, nonassembled row LCRs with LOB columns can be enqueued using the `SET_ENQUEUE_DESTINATION` procedure in the `DBMS_APPLY_ADM` package.

An apply process performs all validation of these requirements. If these requirements are not met, then a row LCR containing LOB columns cannot be applied by an apply process nor processed by an apply handler. In this case, the LCR is moved to the error queue with the rest of the LCRs in the same transaction. For row LCRs with LOB columns, the original, nonassembled row LCRs are placed in the error queue.

See Also:

- *Oracle Streams Concepts and Administration* for more information about apply handlers
- *Oracle Database PL/SQL Packages and Types Reference* for more information about member procedures for row LCRs and for information about the `SET_ENQUEUE_DESTINATION` procedure

Requirements for Rule-Based Transformation Processing of LCRs with LOBs

The following requirements must be met when you are processing row LCRs that contain LOB columns with a rule-based transformation:

- Do not modify LOB column data in a row LCR with a custom rule-based transformation. However, a custom rule-based transformation can modify non-LOB columns in row LCRs that contain LOB columns.
- You cannot use the following row LCR member procedures on a LOB column when you are processing a row LCR with a custom rule-based transformation:
  - `ADD_COLUMN`
  - `SET_LOB_INFORMATION`
  - `SET_LOB_OFFSET`
  - `SET_LOB_OPERATION_SIZE`
  - `SET_VALUE`
  - `SET_VALUES`
- A declarative rule-based transformation created by the `ADD_COLUMN` procedure in the `DBMS_STREAMS_ADM` package cannot add a LOB column to a row LCR.
- Rule-based transformation functions that are run on row LCRs with LOB columns must be deterministic, so that all row LCRs corresponding to the row change are transformed in the same way.
- Do not allow LCRs from a table that contains LOB columns to be processed by an a custom rule-based transformation that is invoked only for specific operations. For example, a custom rule-based transformation that is invoked only for `INSERT` operations should not process LCRs from a table with one or more LOB columns.

Note: If row LCRs contain LOB columns, then rule-based transformations always operate on the original, nonassembled row LCRs.

See Also:

- "Constructing and Enqueuing LCRs" on page 14-2
- *Oracle Streams Concepts and Administration* for information about rule-based transformations
- *Oracle Database PL/SQL Packages and Types Reference* for more information about member procedures for row LCRs
- *Oracle Database SQL Language Reference* for more information about deterministic functions

Managing LCRs Containing LONG or LONG RAW Columns

LONG and LONG RAW data types all can be present in row LCRs captured by a capture process, but these data types are represented by the following data types in row LCRs.

- LONG data type is represented as VARCHAR2 data type in row LCRs.
- LONG RAW data type is represented as RAW data type in row LCRs.

A row change involving a LONG or LONG RAW column can be captured, propagated, and applied as several LCRs. If your environment uses LCRs that contain LONG or LONG RAW columns, then the data portion of the LCR LONG or LONG RAW column must be of type VARCHAR2 or RAW. A VARCHAR2 is interpreted as a LONG, and a RAW is interpreted as a LONG RAW.

You must meet the following requirements when you are processing row LCRs that contain LONG or LONG RAW column data in Oracle Streams:

- Do not modify LONG or LONG RAW column data in an LCR using a custom rule-based transformation. However, you can use a rule-based transformation to modify non LONG and non LONG RAW columns in row LCRs that contain LONG or LONG RAW column data.
- Do not use the SET_VALUE or SET_VALUES row LCR member procedures in a custom rule-based transformation that is processing a row LCR that contains LONG or LONG RAW data. Doing so raises the ORA-26679 error.
- Rule-based transformation functions that are run on LCRs that contain LONG or LONG RAW columns must be deterministic, so that all LCRs corresponding to the row change are transformed in the same way.
- A declarative rule-based transformation created by the ADD_COLUMN procedure in the DBMS_STREAMS_ADM package cannot add a LONG or LONG RAW column to a row LCR.
- You cannot use a procedure DML handler or error handler to process row LCRs that contain LONG or LONG RAW column data.
- Rules used to evaluate LCRs that contain LONG or LONG RAW columns 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.
- You cannot use an apply process to enqueue LCRs that contain LONG or LONG RAW column data into a destination queue. The SET_DESTINATION_QUEUE procedure in the DBMS_APPLY_ADM package sets the destination queue for LCRs that satisfy a specified apply process rule.

Note: LONG and LONG RAW data types cannot be present in row LCRs captured by synchronous captures or constructed by users.

See Also:

- *Oracle Streams Concepts and Administration* for information about rule-based transformations
- *Oracle Database SQL Language Reference* for more information about deterministic functions

Part III

Oracle Streams Replication Best Practices

You can configure Oracle Streams replication in many different ways depending on your business requirements. For example, Oracle Streams can be configured to fulfill the following requirements:

- Replicate data from one database to one or more databases, even if those databases have different structures or naming conventions
- Replicate data between hardware platforms, database releases, and character sets
- Consolidate data from multiple sources with varying structures into a single database
- Provide high availability while performing database or application upgrades or while migrating between hardware platforms

The following chapters in this part describe Oracle Streams replication best practices:

- Chapter 15, "Best Practices for Oracle Streams Replication Databases"
- Chapter 16, "Best Practices for Capture"
- Chapter 17, "Best Practices for Propagation"
- Chapter 18, "Best Practices for Apply"

15

Best Practices for Oracle Streams Replication Databases

An Oracle Streams replication database is a database that participates in an Oracle Streams replication environment. An Oracle Streams replication environment uses Oracle Streams clients to replicate database changes from one database to another. Oracle Streams clients include capture processes, synchronous captures, propagations, and apply processes. This chapter describes general best practices for Oracle Streams replication databases.

This chapter contains these topics:

- Best Practices for Oracle Streams Database Configuration
- Best Practices for Oracle Streams Database Operation
- Best Practices for Oracle Real Application Clusters and Oracle Streams

Best Practices for Oracle Streams Database Configuration

For your Oracle Streams replication environment to run properly and efficiently, follow the best practices in this section when you are configuring the environment. This section contains these topics:

- Use a Separate Queue for Capture and Apply Oracle Streams Clients
- Automate the Oracle Streams Replication Configuration

> **See Also:** Chapter 1, "Preparing for Oracle Streams Replication" describes best practices to follow when preparing for Oracle Streams replication

Use a Separate Queue for Capture and Apply Oracle Streams Clients

Configure a separate queue for each capture process, for each synchronous capture, and for each apply process, and ensure that each queue has its own queue table. Using separate queues is especially important when configuring bidirectional replication between two databases or when a single database receives messages from several other databases.

For example, suppose a database called `db1` is using a capture process to capture changes that will be sent to other databases and is receiving changes from a database named `db2`. The changes received from `db2` are applied by an apply process running on `db1`. In this scenario, create a separate queue for the capture process and apply process at `db1`, and ensure that these queues use different queue tables.

The following example creates the queue for the capture process. The queue name is capture_queue, and this queue uses queue table qt_capture_queue:

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

The following example creates the queue for the apply process. The queue name is apply_queue, and this queue uses queue table qt_apply_queue:

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

Subsequently, specify the queue strmadmin.capture_queue when you configure the capture process at db1, and specify the queue strmadmin.apply_queue when you configure the apply process at db1. If necessary, the SET_UP_QUEUE procedure lets you specify a storage_clause parameter to configure separate tablespace and storage specifications for each queue table.

If you automate the configuration, as described in "Automate the Oracle Streams Replication Configuration" on page 15-2, then each Oracle Streams client is configured with its own queue automatically.

See Also:

- "Creating an ANYDATA Queue" on page 6-1
- "Configuring a Capture Process" on page 5-1
- Chapter 7, "Configuring Implicit Apply"

Automate the Oracle Streams Replication Configuration

Use the following procedures in the DBMS_STREAMS_ADM package to create your Oracle Streams replication environment whenever possible:

- MAINTAIN_GLOBAL configures an Oracle Streams environment that replicates changes at the database level between two databases.
- MAINTAIN_SCHEMAS configures an Oracle Streams environment that replicates changes to specified schemas between two databases.
- MAINTAIN_SIMPLE_TTS clones a simple tablespace from a source database at a destination database and uses Oracle Streams to maintain this tablespace at both databases.
- MAINTAIN_TABLES configures an Oracle Streams environment that replicates changes to specified tables between two databases.
- MAINTAIN_TTS clones a set of tablespaces from a source database at a destination database and uses Oracle Streams to maintain these tablespaces at both databases.
- PRE_INSTANTIATION_SETUP and POST_INSTANTIATION_SETUP configure an Oracle Streams environment that replicates changes either at the database level or to specified tablespaces between two databases. These procedures must be used

together, and instantiation actions must be performed manually, to complete the Oracle Streams replication configuration.

These procedures automate the entire configuration of the Oracle Streams clients at multiple databases. Further, the configuration follows Oracle Streams best practices. For example, these procedures create queue-to-queue propagations whenever possible.

If these procedures are not suitable for your environment, then use the following procedures in the `DBMS_STREAMS_ADM` package to create Oracle Streams clients, rules sets, and rules:

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

These procedures minimize the number of steps required to configure Oracle Streams clients. It is also possible to create rules for nonexistent objects, so ensure that you check the spelling of each object specified in a rule carefully.

Although it is typically not recommended, a propagation or apply process can be used without rule sets or rules if you always want to propagate or apply all of the messages in a queue. However, a capture process requires one or more rule sets with rules, and a synchronous capture requires a positive rule set. You can use the `ADD_GLOBAL_RULES` procedure to capture data manipulation language (DML) changes to an entire database with a capture process if a negative rule set is configured for the capture process to filter out changes to unsupported objects. You can also use the `ADD_GLOBAL_RULES` procedure to capture all data definition language (DDL) changes to the database with a capture process.

The rules in the rule set for a propagation can differ from the rules specified for a capture process or a synchronous capture. For example, to configure that all captured changes be propagated to a destination database, you can run the `ADD_GLOBAL_PROPAGATION_RULES` procedure for the propagation even though multiple rules might have been configured using `ADD_TABLE_RULES` for the capture process or synchronous capture. Similarly, the rules in the rule set for an apply process can differ from the rules specified for the capture process, synchronous capture, and propagation(s) that capture and propagate messages to the apply process.

An Oracle Streams client can process changes for multiple tables or schemas. For the best performance, ensure that the rules for these multiple tables or schemas are simple. Complex rules will impact the performance of Oracle Streams. For example, rules with conditions that include `LIKE` clauses are complex. When you use a procedure in the `DBMS_STREAMS_ADM` package to create rules, the rules are always simple.

When you configure multiple source databases in an Oracle Streams replication environment, change cycling should be avoided. Change cycling means sending a change back to the database where it originated. You can use Oracle Streams tags to prevent change cycling.

See Also:

- "Configuring Replication Using the DBMS_STREAMS_ADM Package" on page 2-3
- "Use Queue-to-Queue Propagations" on page 17-1
- "Oracle Streams Tags in a Replication Environment" on page 10-6 for information about using Oracle Streams tags to avoid change cycling
- *Oracle Streams Concepts and Administration* for more information about simple and complex rules

Best Practices for Oracle Streams Database Operation

After the Oracle Streams replication environment is configured, follow the best practices in this section to keep it running properly and efficiently. This section contains these topics:

- Follow the Best Practices for the Global Name of an Oracle Streams Database
- Monitor Performance and Make Adjustments When Necessary
- Monitor Capture Process's and Synchronous Capture's Queues for Size
- Follow the Oracle Streams Best Practices for Backups
- Adjust the Automatic Collection of Optimizer Statistics
- Check the Alert Log for Oracle Streams Information
- Follow the Best Practices for Removing an Oracle Streams Configuration at a Database

Follow the Best Practices for the Global Name of an Oracle Streams Database

Oracle Streams uses the global name of a database to identify changes from or to a particular database. For example, the system-generated rules for capture, propagation, and apply typically specify the global name of the source database. In addition, changes captured by an Oracle Streams capture process or synchronous capture automatically include the current global name of the source database. If possible, do not modify the global name of a database that is participating in an Oracle Streams replication environment after the environment has been configured. The `GLOBAL_NAMES` initialization parameter must also be set to `TRUE` to guarantee that database link names match the global name of each destination database.

If the global name of an Oracle Streams database must be modified, then do so at a time when no user changes are possible on the database, the queues are empty, and no outstanding changes must be applied by any apply process. When these requirements are met, you can modify the global name of a database and re-create the parts of the Oracle Streams configuration that reference the modified database. All queue subscribers, including propagations and apply processes, must be re-created if the source database global name is changed.

See Also: "Changing the DBID or Global Name of a Source Database" on page 12-23

Monitor Performance and Make Adjustments When Necessary

For Oracle Database 11*g* Release 1 (11.1) and later databases, the Oracle Streams Performance Advisor provides information about how Oracle Streams components are performing. You can use this advisor to monitor the performance of multiple Oracle Streams components in your environment and make adjustments to improve performance when necessary.

The `UTL_SPADV` package also provides performance statistics for an Oracle Streams environment. This package uses the Oracle Streams Performance Advisor to gather performance statistics. The package enables you to format the statistics in output that can be imported into a spreadsheet for analysis.

For databases before Oracle Database 11*g* Release 1 (11.1), you can use `STRMMON` to monitor the performance of an Oracle Streams environment. You can use this tool to obtain a quick overview of the Oracle Streams activity in a database. `STRMMON` reports information in a single line display. You can configure the reporting interval and the number of iterations to display. `STRMMON` is available in the `rdbms/demo` directory in your Oracle home.

See Also:

- *Oracle Streams Concepts and Administration* for information about the Oracle Streams Performance Advisor
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `UTL_SPADV` package

Monitor Capture Process's and Synchronous Capture's Queues for Size

You should monitor the queues used by a capture process to check for queue size. The number of messages in a queue used by a capture process or synchronous capture increases if the messages in the queue cannot be propagated to one or more destination queues. When a source queue becomes large, it often indicates that there is a problem with the Oracle Streams replication environment. Common reasons why messages cannot be propagated include the following:

- One of the destination databases is down for an extended period.
- An apply process at a destination database is disabled for an extended period.
- The queue is the source queue for a propagation that cannot deliver the messages to a particular destination queue for an extended period due to network problems or propagation job problems.

When a capture process's queue becomes large, the capture process pauses for flow control to minimize the number of messages that are spilled to disk. You can monitor the number of messages in a capture process's queue by querying the `V$BUFFERED_QUEUES` dynamic performance view. This view shows the number of messages in memory and the number of messages spilled to disk.

You can monitor the number of messages in a synchronous capture's queue by querying the `V$PERSISTENT_QUEUES` or `V$AQ` dynamic performance view. This view shows the number of messages in different message states in the persistent queue.

Propagation is implemented using Oracle Scheduler. If a job cannot execute 16 successive times, the job is marked as "broken" and is aborted. Check propagation jobs periodically to ensure that they are running successfully to minimize the size of the source queue.

See Also: "Restart Broken Propagations" on page 17-3

Follow the Oracle Streams Best Practices for Backups

The following sections contain information about best practices for backing up source databases and destination databases in an Oracle Streams replication environment. A single database can be both a source database and a destination database.

Best Practices for Backups of an Oracle Streams Source Database

A source database is a database where changes captured by a capture process are generated in a redo log or a database where an Oracle Streams synchronous capture is configured. Follow these best practices for backups of an Oracle Streams source database:

- Use an Oracle Streams tag in the session that runs the online backup SQL statements to ensure that the capture process which captures changes to the source database *will not* capture the backup statements. An online backup statement uses the `BEGIN BACKUP` and `END BACKUP` clauses in an `ALTER TABLESPACE` or `ALTER DATABASE` statement. To set an Oracle Streams session tag, use the `DBMS_STREAMS.SET_TAG` procedure.

  Note: Backups performed using Recovery Manager (RMAN) do not need to set an Oracle Streams session tag.

  See Also: Chapter 10, "Oracle Streams Tags"

- Do not allow any automated backup of the archived logs that might remove archive logs required by a capture process. It is especially important in an Oracle Streams environment that all required archive log files remain available online and in the expected location until the capture process has finished processing them. If a log required by a capture process is unavailable, then the capture process will abort.

  To list each required archive redo log file in a database, run the following query:

```
COLUMN CONSUMER_NAME HEADING 'Capture|Process|Name' FORMAT A15
COLUMN SOURCE_DATABASE HEADING 'Source|Database' FORMAT A10
COLUMN SEQUENCE# HEADING 'Sequence|Number' FORMAT 99999
COLUMN NAME HEADING 'Required|Archived Redo Log|File Name' FORMAT A40

SELECT r.CONSUMER_NAME,
       r.SOURCE_DATABASE,
       r.SEQUENCE#,
       r.NAME
  FROM DBA_REGISTERED_ARCHIVED_LOG r, DBA_CAPTURE c
  WHERE r.CONSUMER_NAME =  c.CAPTURE_NAME AND
        r.NEXT_SCN      >= c.REQUIRED_CHECKPOINT_SCN;
```

- Ensure that all archive log files from all threads are available. Database recovery depends on the availability of these logs, and a missing log will result in incomplete recovery.
- In situations that result in incomplete recovery (point-in-time recovery) at a source database, follow the instructions in "Performing Point-in-Time Recovery on the Source in a Single-Source Environment" on page 12-25 or "Performing Point-in-Time Recovery in a Multiple-Source Environment" on page 12-29.

Best Practices for Backups of an Oracle Streams Destination Database

In an Oracle Streams replication environment, a destination database is a database where an apply process applies changes. Follow these best practices for backups of an Oracle Streams destination database:

- Ensure that the `commit_serialization` apply process parameter is set to `FULL`.
- In situations that result in incomplete recovery (point-in-time recovery) at a destination database, follow the instructions in "Performing Point-in-Time Recovery on a Destination Database" on page 12-30.

Adjust the Automatic Collection of Optimizer Statistics

Every night by default, the optimizer automatically collects statistics on tables whose statistics have become stale. For volatile tables, such as Oracle Streams queue tables, it is likely that the statistics collection job runs when these tables might not have data that is representative of their full load period.

You create these volatile queue tables using the `DBMS_AQADM.CREATE_QUEUE_TABLE` or `DBMS_STREAMS_ADM.SETUP_QUEUE` procedure. You specify the queue table name when you run these procedures. In addition to the queue table, the following tables are created when the queue table is created and are also volatile:

- `AQ$_`*`queue_table_name`*`_I`
- `AQ$_`*`queue_table_name`*`_H`
- `AQ$_`*`queue_table_name`*`_T`
- `AQ$_`*`queue_table_name`*`_P`
- `AQ$_`*`queue_table_name`*`_D`
- `AQ$_`*`queue_table_name`*`_C`

Replace *`queue_table_name`* with the name of the queue table.

Oracle recommends that you collect statistics on volatile tables by completing the following steps:

1. Run the `DBMS_STATS.GATHER_TABLE_STATS` procedure manually on volatile tables when these tables are at their fullest.
2. Immediately after the statistics are collected on volatile tables, run the `DBMS_STATS.LOCK_TABLE_STATS` procedure on these tables.

Locking the statistics on volatile tables ensures that the automatic statistics collection job skips these tables, and the tables are not analyzed.

> **See Also:** *Oracle Database Performance Tuning Guide* for more information about managing optimizer statistics

Check the Alert Log for Oracle Streams Information

By default, the alert log contains information about why Oracle Streams capture and apply processes stopped. Also, Oracle Streams capture and apply processes report long-running and large transactions in the alert log.

Long-running transactions are open transactions with no activity (that is, no new change records, rollbacks, or commits) for an extended period (20 minutes). Large transactions are open transactions with a large number of change records. The alert log reports whether a long-running or large transaction has been seen every 20 minutes. Not all such transactions are reported, because only one transaction is reported for

each 20 minute period. When the commit or rollback is received, this information is reported in the alert log as well.

You can use the following views for information about long-running transactions:

- The `V$STREAMS_TRANSACTION` dynamic performance view enables monitoring of long running transactions that are currently being processed by Oracle Streams capture processes and apply processes.
- The `DBA_APPLY_SPILL_TXN` and `V$STREAMS_APPLY_READER` views enable you to monitor the number of transactions and messages spilled by an apply process.

Note: The `V$STREAMS_TRANSACTION` view does not pertain to synchronous captures.

See Also: *Oracle Streams Concepts and Administration* for more information about Oracle Streams information in the alert log

Follow the Best Practices for Removing an Oracle Streams Configuration at a Database

To completely remove the Oracle Streams configuration at a database, complete the following steps:

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

   See *Oracle Database Administrator's Guide* for instructions about connecting to a database in SQL*Plus.

2. Run the `DBMS_STREAMS_ADM.REMOVE_STREAMS_CONFIGURATION` procedure.
3. Drop the Oracle Streams administrator, if possible.

Best Practices for Oracle Real Application Clusters and Oracle Streams

The following best practices are for Oracle Real Application Clusters (Oracle RAC) databases in Oracle Streams replication environments:

- Make Archive Log Files of All Threads Available to Capture Processes
- Follow the Best Practices for the Global Name of an Oracle RAC Database
- Follow the Best Practices for Configuring and Managing Propagations
- Follow the Best Practices for Queue Ownership

See Also: *Oracle Streams Concepts and Administration* for more information about how Oracle Streams works with Oracle RAC

Make Archive Log Files of All Threads Available to Capture Processes

The archive log files of all threads from all instances must be available to any instance running a capture process. This requirement pertains to both local and downstream capture processes.

Follow the Best Practices for the Global Name of an Oracle RAC Database

The general best practices described in "Follow the Best Practices for the Global Name of an Oracle Streams Database" on page 15-4 also apply to Oracle Real Application Clusters (Oracle RAC) databases in an Oracle Streams environment. In addition, if the

global name of an Oracle RAC destination database does not match the *DB_NAME.DB_DOMAIN* of the database, then include the global name for the database in the list of services for the database specified by the SERVICE_NAMES initialization parameter.

In the tnsnames.ora file, ensure that the CONNECT_DATA clause in the connect descriptor specifies the global name of the destination database for the SERVICE_NAME. Also, ensure that the CONNECT_DATA clause does not include the INSTANCE_NAME parameter.

If the global name of an Oracle RAC database that contains Oracle Streams propagations is changed, then drop and re-create all propagations. Ensure that the new propagations are queue-to-queue propagations by setting the queue_to_queue parameter set to TRUE during creation.

If the global name of an Oracle RAC destination database must be changed, then ensure that the queue used by each apply process is empty and that there are no unapplied transactions before changing the global name. After the global name is changed, drop and re-create each apply process's queue and each apply process.

> **See Also:** "Follow the Best Practices for Queue Ownership" on page 15-9 for more information about the SERVICE_NAME parameter in the tnsnames.ora file

Follow the Best Practices for Configuring and Managing Propagations

The general best practices described in "Restart Broken Propagations" on page 17-3 also apply to Oracle Real Application Clusters (Oracle RAC) databases in an Oracle Streams environment. Use the procedures START_PROPAGATION and STOP_PROPAGATION in the DBMS_PROPAGATION_ADM package to start and stop propagations. These procedures automatically handle queue-to-queue propagation.

Also, on an Oracle RAC database, a service is created for each buffered queue. This service always runs on the owner instance of the destination queue and follows the ownership of this queue upon queue ownership switches, which include instance startup, instance shutdown, and so on. This service is used by queue-to-queue propagations. You can query NETWORK_NAME column of the DBA_SERVICES data dictionary view to determine the service name for a queue-to-queue propagation. If you are running Oracle RAC instances, and you have queues that were created before Oracle Database 10*g* Release 2, then drop and re-create these queues to take advantage of the automatic service generation and queue-to-queue propagation. Ensure that you re-create these queues when they are empty and no new messages are being enqueued into them.

> **See Also:** "Use Queue-to-Queue Propagations" on page 17-1

Follow the Best Practices for Queue Ownership

All Oracle Streams processing is done at the owning instance of the queue used by the Oracle Streams client. To determine the owning instance of each ANYDATA queue in a database, run the following query:

```
SELECT q.OWNER, q.NAME, t.QUEUE_TABLE, t.OWNER_INSTANCE
   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;
```

When Oracle Streams is configured in an Oracle Real Application Clusters (Oracle RAC) environment, each queue table has an owning instance. Also, all queues within

an individual queue table are owned by the same instance. The following Oracle Streams clients use the owning instance of the relevant queue to perform their work:

- Each capture process is run at the owning instance of its queue.
- Each propagation is run at the owning instance of the propagation's source queue.
- Each propagation must connect to the owning instance of the propagation's destination queue.
- Each apply process is run at the owning instance of its queue.

You can configure ownership of a queue to remain on a specific instance, if that instance is available, by running the `DBMS_AQADM.ALTER_QUEUE_TABLE` procedure and setting the `primary_instance` and `secondary_instance` parameters. When the primary instance of a queue table is set to a specific instance, the queue ownership will return to the specified instance whenever the instance is running.

Capture processes and apply processes automatically follow the ownership of the queue. If the ownership changes while process is running, then the process stops on the current instance and restarts on the new owner instance.

Queue-to-queue propagations send messages only to the specific queue identified as the destination queue. Also, the source database link for the destination database connect descriptor must specify the correct service to connect to the destination database. The `CONNECT_DATA` clause in the connect descriptor should specify the global name of the destination database for the `SERVICE_NAME`.

For example, consider the `tnsnames.ora` file for a database with the global name `db.mycompany.com`. Assume that the alias name for the first instance is `db1` and that the alias for the second instance is `db2`. The `tnsnames.ora` file for this database might include the following entries:

```
db.mycompany.com=
 (description=
  (load_balance=on)
   (address=(protocol=tcp)(host=node1-vip)(port=1521))
   (address=(protocol=tcp)(host=node2-vip)(port=1521))
  (connect_data=
     (service_name=db.mycompany.com)))

db1.mycompany.com=
 (description=
  (address=(protocol=tcp)(host=node1-vip)(port=1521))
  (connect_data=
    (service_name=db.mycompany.com)
    (instance_name=db1)))

db2.mycompany.com=
 (description=
  (address=(protocol=tcp)(host=node2-vip)(port=1521))
  (connect_data=
    (service_name=db.mycompany.com)
    (instance_name=db2)))
```

16

Best Practices for Capture

This chapter describes the best practices for capturing changes with a capture process or a synchronous capture in an Oracle Streams replication environment. This chapter includes these topics:

- Best Practices for Capture Process Configuration
- Best Practices for Capture Process Operation
- Best Practices for Synchronous Capture Configuration

See Also:

- "Best Practices for Oracle Real Application Clusters and Oracle Streams" on page 15-8
- Chapter 1, "Preparing for Oracle Streams Replication" describes best practices to follow when preparing for Oracle Streams capture processes

Best Practices for Capture Process Configuration

The following sections describe best practices for configuring capture processes:

- Grant the Required Privileges to the Capture User
- Set Capture Process Parallelism
- Set the Checkpoint Retention Time

Grant the Required Privileges to the Capture User

The capture user is the user in whose security domain a capture process captures changes that satisfy its rule sets and runs custom rule-based transformations configured for capture process rules.

The capture user for a capture process is configured when you create a capture process, and the capture user can be modified when you alter a capture process. Grant the following privileges to the apply user:

- `EXECUTE` privilege on the rule sets used by the capture process
- `EXECUTE` privilege on all rule-based transformation functions used in the positive rule set

These privileges can be granted directly to the capture user, or they can be granted through roles.

In addition, the capture user must be granted `EXECUTE` privilege on all packages, including Oracle-supplied packages, that are invoked in rule-based transformations run by the capture process. These privileges must be granted directly to the capture user. They cannot be granted through roles.

See Also:

- *Oracle Database Security Guide* for general information about granting privileges
- *Oracle Streams Concepts and Administration* for information about granting privileges on rule sets

Set Capture Process Parallelism

Set the parallelism of each capture process by specifying the `parallelism` parameter in the `DBMS_CAPTURE_ADM.SET_PARAMETER` procedure. The `parallelism` parameter controls the number of processes that concurrently mine the redo log for changes.

The default setting for the `parallelism` capture process parameter is `1`, and the default `parallelism` setting is appropriate for most capture process configurations. Ensure that the `PROCESSES` initialization parameter is set appropriately when you set the `parallelism` capture process parameter.

See Also:

- *Oracle Streams Concepts and Administration*
- *Oracle Database PL/SQL Packages and Types Reference*

Set the Checkpoint Retention Time

Set the checkpoint retention time for each capture process. Periodically, a capture process takes a checkpoint to facilitate quicker restart. These checkpoints are maintained in the `SYSAUX` tablespace by default. The checkpoint retention time for a capture process controls the amount of checkpoint data it retains. The checkpoint retention time specifies the number of days before the required checkpoint SCN to retain checkpoints. When a checkpoint is older than the specified time period, the capture process purges the checkpoint.

When checkpoints are purged, the first SCN for the capture process moves forward, and Oracle Database writes a message including the text "first scn changed" to the alert log. The first SCN is the lowest possible SCN available for capturing changes. The checkpoint retention time is set when you create a capture process, and it can be set when you alter a capture process. When the checkpoint retention time is exceeded, the first SCN is moved forward, and the Oracle Streams metadata tables before this new first SCN are purged. The space used by these tables in the `SYSAUX` tablespace is reclaimed. To alter the checkpoint retention time for a capture process, use the `ALTER_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package and specify the new retention time with the `checkpoint_retention_time` parameter.

The default value for the checkpoint retention time is `60` days. If checkpoints are available for a time in the past, then the capture process can recapture changes to recover a destination database. You should set the checkpoint retention time to an appropriate value for your environment. A typical setting is `7` days.

See Also:

- *Oracle Streams Concepts and Administration*
- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `ALTER_CAPTURE` procedure

Best Practices for Capture Process Operation

The following sections describe best practices for operating existing capture processes:

- Configure a Heartbeat Table at Each Source Database in an Oracle Streams Environment
- Perform a Dictionary Build and Prepare Database Objects for Instantiation Periodically
- Minimize the Performance Impact of Batch Processing

Configure a Heartbeat Table at Each Source Database in an Oracle Streams Environment

You can use a heartbeat table to ensure that changes are being replicated in an Oracle Streams replication environment. Specifically, you can check the `APPLIED_SCN` value in the `DBA_CAPTURE` data dictionary view at the capture database to ensure that it is updated periodically. A heartbeat table is especially useful for databases that have a low activity rate because you can ensure that the replication environment is working properly even if there are few replicated changes.

An Oracle Streams capture process requests a checkpoint after every 10 MB of generated redo. During the checkpoint, the metadata for Oracle Streams is maintained if there are active transactions. Implementing a heartbeat table ensures that there are open transactions occurring regularly in the source database, thereby providing additional opportunities for the metadata to be updated frequently. Additionally, the heartbeat table provides quick feedback to the database administrator about the health of the Oracle Streams replication environment.

To implement a heartbeat table, complete the following steps:

1. Create a table at the source database that includes a date or time stamp column and the global name of the source database.
2. Instantiate the table at the destination database. If there are multiple destination databases, then instantiate the heartbeat table at each destination database.
3. Add a rule to the positive rule set for the capture process that captures changes to the source database. The rule instructs the capture process to capture changes to the heartbeat table.
4. Add a rule to the positive rule set for the propagation that propagates changes from the source database to the destination database. The rule instructs the propagation to propagate LCRs for the heartbeat table. If there are multiple propagations, then add the rule to the rule set for each propagation. If your environment uses directed networks, then you might need to add rules to propagations at several databases.
5. Add a rule to the positive rule set for the apply process that applies changes that originated at the source database. The rule instructs the apply process to apply changes to the heartbeat table. If there are multiple apply processes at multiple databases that apply the changes that originated at the source database, then add a rule to each the apply process.

6. Configure an automated job to update the heartbeat table at the source database periodically. For example, the table might be updated every minute.
7. Monitor the Oracle Streams replication environment to verify that changes to the heartbeat table at the source database are being replicated to the destination database.

Perform a Dictionary Build and Prepare Database Objects for Instantiation Periodically

Perform a data dictionary build in the source database redo periodically. Run the `DBMS_CAPTURE_ADM.BUILD` procedure to build a current copy of the data dictionary in the redo log. Ideally, database objects should be prepared for instantiation after a build is performed. Run one or more of the following procedures in the `DBMS_CAPTURE_ADM` package to prepare database objects for instantiation:

- `PREPARE_GLOBAL_INSTANTIATION`
- `PREPARE_SCHEMA_INSTANTIATION`
- `PREPARE_TABLE_INSTANTIATION`

Each of the database objects for which a capture process captures changes should be prepared for instantiation periodically. You can reduce the amount of redo data that must be processed if additional capture process are created or if an existing capture process must be re-created by performing a build and preparing shared objects for instantiation periodically.

See Also:

- *Oracle Streams Concepts and Administration*
- "Capture Rules and Preparation for Instantiation" on page 8-3

Minimize the Performance Impact of Batch Processing

For best performance, the commit point for a batch processing job should be kept low. Also, if a large batch processing job must be run at a source database, then consider running it at each Oracle Streams replication database independently. If this technique is used, then ensure that the changes resulting from the batch processing job are not replicated. To accomplish this, run the `DBMS_STREAMS.SET_TAG` procedure in the session that runs the batch processing job, and set the session tag to a value that *will not* be captured by a capture process.

See Also: Chapter 10, "Oracle Streams Tags"

Best Practices for Synchronous Capture Configuration

Creating and managing a synchronous capture is simplified when you use the `DBMS_STREAMS_ADM` package. Specifically, use the following procedures in the `DBMS_STREAMS_ADM` package to create a synchronous capture and configure synchronous capture rules:

- `ADD_TABLE_RULES`
- `ADD_SUBSET_RULES`

Also, use the `REMOVE_RULE` procedure in the `DBMS_STREAMS_ADM` package to remove a rule from a synchronous capture rule set or to drop a rule in a synchronous capture rule set.

See Also: "Configuring Synchronous Capture" on page 5-17

17

Best Practices for Propagation

This chapter describes the best practices for propagating messages in an Oracle Streams replication environment. This chapter includes these topics:

- Best Practices for Propagation Configuration
- Best Practices for Propagation Operation

See Also: "Best Practices for Oracle Real Application Clusters and Oracle Streams" on page 15-8

Best Practices for Propagation Configuration

This following sections describe best practices for configuring propagations:

- Use Queue-to-Queue Propagations
- Set the Propagation Latency for Each Propagation
- Increase the SDU in a Wide Area Network for Better Network Performance

Use Queue-to-Queue Propagations

A propagation can be queue-to-queue or queue-to-database link (queue-to-dblink). Use queue-to-queue propagations whenever possible. A queue-to-queue propagation always has its own exclusive propagation job to propagate messages from the source queue to the destination queue. Because each propagation job has its own propagation schedule, the propagation schedule of each queue-to-queue propagation can be managed separately. Even when multiple queue-to-queue propagations use the same database link, you can enable, disable, or set the propagation schedule for each queue-to-queue propagation separately.

Propagations configured before Oracle Database 10*g* Release 2 are queue-to-dblink propagations. Also, any propagation that includes a queue in a database before Oracle Database 10*g* Release 2 must be a queue-to-dblink propagation. When queue-to-dblink propagations are used, propagation will not succeed if the database link no longer connects to the owning instance of the destination queue.

If you have queue-to-dblink propagations created in a prior release of Oracle Database, you can re-create these propagation during a maintenance window to use queue-to-queue propagation. To re-create a propagation, complete the following steps:

1. Stop the propagation.
2. Ensure that the source queue is empty.
3. Ensure that the destination queue is empty and has no unapplied, spilled messages before you drop the propagation.

4. Re-create the propagation with the `queue_to_queue` parameter set to `TRUE` in the creation procedure.

If you automate the configuration, as described in "Automate the Oracle Streams Replication Configuration" on page 15-2, then each propagation uses queue-to-queue propagation automatically.

See Also:

- "Follow the Best Practices for Configuring and Managing Propagations" on page 15-9 for information about propagations in an Oracle Real Application Clusters (Oracle RAC) environment
- "Creating Oracle Streams Propagations Between ANYDATA Queues" on page 6-3

Set the Propagation Latency for Each Propagation

Propagation latency is the maximum wait, in seconds, in the propagation window for a message to be propagated after it is enqueued. Set the propagation latency to an appropriate value for each propagation in your Oracle Streams replication environment. The default propagation latency value is 60.

The following scenarios describe how a propagation will behave given different propagation latency values:

- If latency is set to 60, and there are no messages enqueued during the propagation window, then the queue will not be checked for 60 seconds for messages to be propagated to the specified destination. That is, messages enqueued into the queue for the propagation destination will not be propagated for at least 60 more seconds.
- If the latency is set to 600, and there are no messages enqueued during the propagation window, then the queue will not be checked for 10 minutes for messages to be propagated to the specified destination. That is, messages enqueued into the queue for the propagation destination will not be propagated for at least 10 more minutes.
- If the latency is set to 0, then a job slave will be waiting for messages to be enqueued for the destination and, as soon as a message is enqueued, it will be propagated. Setting latency to 0 is not recommended because it might affect the ability of the database to shut down in `NORMAL` mode.

You can set propagation parameters using the `ALTER_PROPAGATION_SCHEDULE` procedure from the `DBMS_AQADM` package. For example, to set the latency of a propagation from the `q1` source queue owned by `strmadmin` to the destination queue `q2` at the database with a global name of `dbs2.example.com`, log in as the Oracle Streams administrator, and run the following procedure:

```
BEGIN
  DBMS_AQADM.ALTER_PROPAGATION_SCHEDULE(
   queue_name        => 'strmadmin.q1',
   destination       => 'dbs2.example.com',
   latency           => 5,
   destination_queue => 'strmadmin.q2');
END;
/
```

Note: If the `latency` parameter is not specified, then propagation latency is set to the default value of 60.

See Also: *Oracle Streams Concepts and Administration*

Increase the SDU in a Wide Area Network for Better Network Performance

When using Oracle Streams propagation in a Wide Area Network (WAN), increase the session data unit (SDU) to improve the propagation performance. The maximum value for SDU is 32K (32767). The SDU value for network transmission is negotiated between the sender and receiver sides of the connection, and the minimum SDU value of the two endpoints is used for any individual connection. To take advantage of an increased SDU for Oracle Streams propagation, the receiving side `sqlnet.ora` file must include the `DEFAULT_SDU_SIZE` parameter. The receiving side `listener.ora` file must indicate the SDU change for the system identifier (SID). The sending side `tnsnames.ora` file connect string must also include the SDU modification for the particular service.

In addition, the `SEND_BUF_SIZE` and `RECV_BUF_SIZE` parameters in the `listener.ora` and `tnsnames.ora` files increase the performance of propagation on your system. These parameters increase the size of the buffer used to send or receive the propagated messages. These parameters should only be increased after careful analysis of their overall impact on system performance.

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

Best Practices for Propagation Operation

The following section describes best practices for operating existing propagations.

Restart Broken Propagations

Sometimes, the propagation job for a propagation might become "broken" or fail to start after it encounters an error or after a database restart. Typically, stopping and restarting the propagation solves the problem. For example, to stop and restart a propagation named `prop1`, run the following procedures:

```
BEGIN
  DBMS_PROPAGATION_ADM.STOP_PROPAGATION(
    propagation_name => 'prop1');
END;
/

BEGIN
  DBMS_PROPAGATION_ADM.START_PROPAGATION(
    propagation_name => 'prop1');
END;
/
```

If running these procedures does not correct the problem, then run the `STOP_PROPAGATION` procedure with the `force` parameter set to `TRUE`, and restart propagation. For example:

```
BEGIN
  DBMS_PROPAGATION_ADM.STOP_PROPAGATION(
    propagation_name => 'prop1',
    force            => TRUE);
```

```
END;
/

BEGIN
  DBMS_PROPAGATION_ADM.START_PROPAGATION(
    propagation_name => 'prop1');
END;
/
```

When you stop a propagation with the `force` parameter set to `TRUE`, the statistics for the propagation are cleared.

18

Best Practices for Apply

This chapter describes the best practices for applying changes with an apply process in an Oracle Streams replication environment. This chapter includes these topics:

- Best Practices for Destination Database Configuration
- Best Practices for Apply Process Configuration
- Best Practices for Apply Process Operation

See Also: "Best Practices for Oracle Real Application Clusters and Oracle Streams" on page 15-8

Best Practices for Destination Database Configuration

In an Oracle Streams replication environment, a destination database is a database where an apply process applies changes. This section contains these topics:

- Grant Required Privileges to the Apply User
- Set Instantiation SCN Values
- Configure Conflict Resolution

Grant Required Privileges to the Apply User

The apply user is the user in whose security domain an apply process performs the following actions:

- Dequeues messages that satisfy its rule sets
- Runs custom rule-based transformations configured for apply process rules
- Applies messages directly to database objects
- Runs apply handlers configured for the apply process

The apply user for an apply process is configured when you create an apply process, and the apply user can be modified when you alter an apply process. Grant the following privileges to the apply user:

- If the apply process applies data manipulation language (DML) changes to a table, then grant `INSERT`, `UPDATE`, and `DELETE` privileges on the table to the apply user.
- If the apply process applies data definition language (DDL) changes to a table, then grant `CREATE TABLE` or `CREATE ANY TABLE`, `CREAT INDEX` or `CREATE ANY INDEX`, and `CREATE PROCEDURE` or `CREATE ANY PROCEDURE` to the apply user.
- Grant `EXECUTE` privilege on the rule sets used by the apply process.

- Grant EXECUTE privilege on all custom rule-based transformation functions specified for rules in the positive rule set of the apply process.
- Grant EXECUTE privilege on any apply handlers used by the apply process.
- Grant privileges to dequeue messages from the apply process's queue.

These privileges can be granted directly to the apply user, or they can be granted through roles.

In addition, the apply user must be granted EXECUTE privilege on all packages, including Oracle-supplied packages, that are invoked in subprograms run by the apply process. These privileges must be granted directly to the apply user. They cannot be granted through roles.

See Also:

- *Oracle Database Security Guide* for general information about granting privileges
- *Oracle Streams Concepts and Administration* for information about granting privileges on rule sets

Set Instantiation SCN Values

An instantiation SCN value must be set for each database object to which an apply process applies changes. Confirm that an instantiation SCN is set for all such objects, and set any required instantiation SCN values that are not set.

Instantiation SCN values can be set in various ways, including during export/import, by Recovery Manager (RMAN), or manually. To set instantiation SCN values manually, use one of the following procedures in the DBMS_APPLY_ADM package:

- SET_TABLE_INSTANTIATION_SCN
- SET_SCHEMA_INSTANTIATION_SCN
- SET_GLOBAL_INSTANTIATION_SCN

For example, to set the instantiation SCN manually for each table in the a schema, run the SET_SCHEMA_INSTANTIATION_SCN procedure with the recursive parameter set to TRUE. If an apply process applies data definition language (DDL) changes, then set the instantiation SCN values at a level higher than table level using either the SET_SCHEMA_INSTANTIATION_SCN or SET_GLOBAL_INSTANTIATION_SCN procedure.

See Also: Chapter 8, "Instantiation and Oracle Streams Replication" for more information about instantiation and setting instantiation SCN values

Configure Conflict Resolution

If updates will be performed at multiple databases for the same shared database object, then ensure that you configure conflict resolution for that object. To simplify conflict resolution for tables with LOB columns, create an error handler to handle errors for the table. When registering the error handler using the DBMS_APPLY_ADM.SET_DML_HANDLER procedure, ensure that you set the assemble_lobs parameter to TRUE.

If you configure conflict resolution at a destination database, then additional supplemental logging is required at the source database. Specifically, 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.

See Also:

- Chapter 9, "Oracle Streams Conflict Resolution"
- "Specifying Supplemental Logging" on page 1-28

Best Practices for Apply Process Configuration

The following sections describe best practices for configuring apply processes:

- Set Apply Process Parallelism
- Consider Allowing Apply Processes to Continue When They Encounter Errors

Set Apply Process Parallelism

Set the parallelism of an apply process by specifying the `parallelism` parameter in the `DBMS_APPLY_ADM.SET_PARAMETER` procedure. The `parallelism` parameter controls the number of processes that concurrently apply changes. The default setting for the `parallelism` apply process parameter is `4`.

Typically, apply process parallelism is set to either `1`, `4`, `8`, or `16`. The setting that is best for a particular apply process depends on the load applied and the processing power of the computer system running the database. Follow these guidelines when setting apply process parallelism:

- If the load is heavy for the apply process and the computer system running the database has excellent processing power, then set apply process parallelism to `8` or `16`. Multiple high-speed CPUs provide excellent processing power.
- If the is light for the apply process, then set apply process parallelism to `1` or `4`.
- If the computer system running the database has less than optimal processing power, then set apply process parallelism to `1` or `4`.

Ensure that the `PROCESSES` initialization parameter is set appropriately when you set the `parallelism` apply process parameter.

In addition, if parallelism is greater than 1 for an apply process, then ensure that any database objects whose changes are applied by the apply process have the appropriate settings for the `INITRANS` and `PCTFREE` parameters. The `INITRANS` parameter specifies the initial number of concurrent transaction entries allocated within each data block allocated to the database object. Set the `INITRANS` parameter to the parallelism of the apply process or higher. The `PCTFREE` parameter specifies the percentage of space in each data block of the database object reserved for future updates to rows of the object. The `PCTFREE` parameter should be set to 10 or higher. You can modify these parameters for a table using the `ALTER TABLE` statement

See Also:

- *Oracle Streams Concepts and Administration*
- *Oracle Database PL/SQL Packages and Types Reference*
- *Oracle Database SQL Language Reference* for more information about the `ALTER TABLE` statement

Consider Allowing Apply Processes to Continue When They Encounter Errors

When the `disable_on_error` apply process parameter is set to `Y`, the apply process is disabled on the first unresolved error, even if the error is not irrecoverable. When the `disable_on_error` apply process parameter is set to `N`, the apply process continues regardless of unresolved errors. The default setting for this parameter is `Y`. If you do not want an apply process to become disabled when it encounters errors, then set the `disable_on_error` parameter to `N`.

Best Practices for Apply Process Operation

The following section describes best practices for operating existing apply processes.

Manage Apply Errors

The error queue contains all of the current apply errors for a database. If there are multiple apply processes in a database, then the error queue contains the apply errors for each apply process. If an apply process encounters an error when it tries to apply a logical change record (LCR) in a transaction, then all of the LCRs in the transaction are moved to the error queue. To view information about apply errors, query the `DBA_APPLY_ERROR` data dictionary view or use Enterprise Manager.

The `MESSAGE_NUMBER` column in the `DBA_APPLY_ERROR` view indicates the LCR within the transaction that resulted in the error. When apply errors are encountered, correct the problem(s) that caused the error(s), and either retry or delete the error transaction in the error queue.

> **See Also:** *Oracle Streams Concepts and Administration* for information about managing apply errors and for information about displaying detailed information for the column values of each LCR in an error transaction

Part IV

Appendixes

This part includes the following appendix:

- Appendix A, "Migrating Advanced Replication to Oracle Streams"

A

Migrating Advanced Replication to Oracle Streams

Database administrators who have been using Advanced Replication to maintain replicated database objects at different sites can migrate their Advanced Replication environment to an Oracle Streams environment. This chapter provides a conceptual overview of the steps in this process and documents each step with procedures and examples.

This chapter contains these topics:

- Overview of the Migration Process
- Preparing to Generate the Migration Script
- Generating and Modifying the Migration Script
- Performing the Migration for Advanced Replication to Oracle Streams
- Re-creating Master Sites to Retain Materialized View Groups

---

Note: The example of a generated migration script is not included in the PDF version of this chapter, but it is included in the HTML version of the chapter.

See Also: *Oracle Database Advanced Replication* and *Oracle Database Advanced Replication Management API Reference* for more information about Advanced Replication

Overview of the Migration Process

The following sections provide a conceptual overview of the migration process:

- Migration Script Generation and Use
- Modification of the Migration Script
- Actions Performed by the Generated Script
- Migration Script Errors
- Manual Migration of Updatable Materialized Views
- Advanced Replication Elements that Cannot Be Migrated to Oracle Streams

Migration Script Generation and Use

You can use the procedure DBMS_REPCAT.STREAMS_MIGRATION to generate a SQL*Plus script that migrates an existing Advanced Replication environment to an Oracle Streams environment. When you run the DBMS_REPCAT.STREAMS_MIGRATION procedure at a master definition site in a multimaster replication environment, it generates a SQL*Plus script in a file at a location that you specify. Once the script is generated, you run it at each master site in your Advanced Replication environment to set up an Oracle Streams environment for each master site. To successfully generate the Oracle Streams environment for your replication groups, the replication groups for which you run the script must have the same master sites. If replication groups have different master sites, then you can generate multiple scripts to migrate each replication group to Oracle Streams.

At times, you must stop, or quiesce, all replication activity for a replication group so that you can perform certain administrative tasks. You do not need to quiesce the replication groups when you run the DBMS_REPCAT.STREAMS_MIGRATION procedure. However, you must quiesce the replication groups being migrated to Oracle Streams when you run the generated script at the master sites. Because you have quiesced the replication groups to run the script at the master sites, you do not have to stop any existing capture processes, propagation jobs, or apply processes at these sites.

Modification of the Migration Script

The generated migration script uses comments to indicate Advanced Replication elements that cannot be converted to Oracle Streams. It also provides suggestions for modifying the script to convert these elements to Oracle Streams. You can use these suggestions to edit the script before you run it. You can also customize the migration script in other ways to meet your needs.

The script sets all parameters when it runs PL/SQL procedures and functions. When you generate the script, it sets default values for parameters that typically do not need to be changed. However, you can change these default parameters by editing the script if necessary. The parameters with default settings include the following:

- include_dml
- include_ddl
- include_tagged_lcr

The beginning of the script has a list of variables for names that are used by the procedures and functions in the script. When you generate the script, it sets these variables to default values that you should not need to change. However, you can change the default settings for these variables if necessary. The variables specify names of queues, capture processes, propagations, and apply processes.

Actions Performed by the Generated Script

The migration script performs the following actions:

- Prints warnings in comments if the replication groups contain features that cannot be converted to Oracle Streams.
- Creates ANYDATA queues, if needed, using the DBMS_STREAMS_ADM.SET_UP_QUEUE procedure.
- Configures propagation between all master sites using the DBMS_STREAMS_ADMIN.ADD_TABLE_PROPAGATION_RULES procedure for each table.

- Configures capture at each master site using the `DBMS_STREAMS_ADMIN.ADD_TABLE_RULES` procedure for each table.
- Configures apply for changes from all the other master sites using the `DBMS_STREAMS_ADMIN.ADD_TABLE_RULES` procedure for each table.
- Sets the instantiation SCN for each replicated object at each site where changes to the object are applied.
- Creates the necessary supplemental log groups at source databases.
- Sets key columns, if any.
- Configures conflict resolution if it was configured for the Advanced Replication environment being migrated.

Migration Script Errors

If Oracle encounters an error while running the migration script, then the migration script exits immediately. If this happens, then you must modify the script to run any commands that have not already been executed successfully.

Manual Migration of Updatable Materialized Views

You cannot migrate updatable materialized views using the migration script. You must migrate updatable materialized views from an Advanced Replication environment to an Oracle Streams environment manually.

> **See Also:** "Re-creating Master Sites to Retain Materialized View Groups" on page A-12

Advanced Replication Elements that Cannot Be Migrated to Oracle Streams

Oracle Streams does not support the following:

- Replication of changes to tables with columns of the following data types: `BFILE`, `ROWID`, and user-defined types (including object types, `REF`s, varrays, and nested tables)
- Synchronous replication

If your current Advanced Replication environment uses these features, then these elements of the environment cannot be migrated to Oracle Streams. In this case, you might decide not to migrate the environment to Oracle Streams now, or you might decide to modify the environment so that it can be migrated to Oracle Streams.

Preparing to Generate the Migration Script

Before generating the migration script, ensure that all the following conditions are met:

- All the replication groups must have the same master site(s).
- The master site that generates the migration script must be running Oracle Database 10*g* or later.
- The other master sites that run the script, but do not generate the script, must be running Oracle9*i* Database Release 2 (9.2) or later.

Generating and Modifying the Migration Script

To generate the migration script, use the procedure `DBMS_REPCAT.STREAMS_MIGRATION` in the `DBMS_REPCAT` package. The syntax for this procedure is as follows:

```
DBMS_REPCAT.STREAMS_MIGRATION (
     gnames              IN   DBMS_UTILITY.NAME_ARRAY,
     file_location       IN   VARCHAR2,
     filename            IN   VARCHAR2);
```

Parameters for the `DBMS_REPCAT.STREAMS_MIGRATION` procedure include the following:

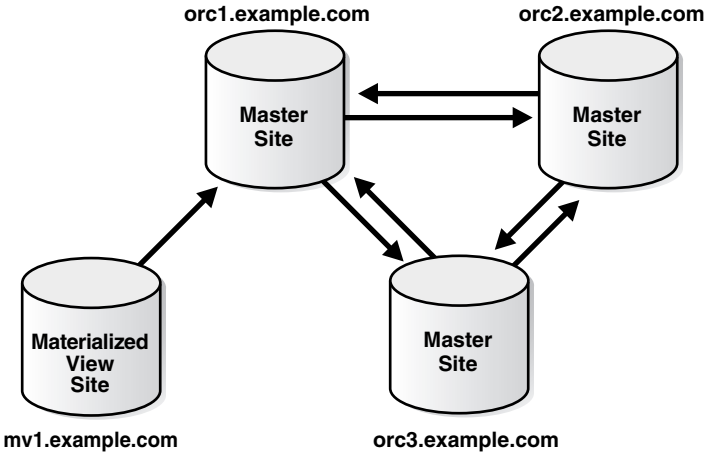
- `gnames`: List of replication groups to migrate to Oracle Streams. The replication groups listed must all contain the same master sites. An error is raised if the replication groups have different masters.
- `file_location`: Directory location of the migration script.
- `filename`: Name of the migration script.

This procedure generates a script for setting up an Oracle Streams environment for the given replication groups. The script can be customized and run at each master site.

Example Advanced Replication Environment to be Migrated to Oracle Streams

Figure A–1 shows the Advanced Replication environment that will be migrated to Oracle Streams in this example.

Figure A–1 Advanced Replication Environment to be Migrated to Oracle Streams



This Advanced Replication environment has the following characteristics:

- The `orc1.example.com` database is the master definition site for a three-way master configuration that also includes `orc2.example.com` and `orc3.example.com`.
- The `orc1.example.com` database is the master site for the `mv1.example.com` materialized view site.
- The environment replicates changes to the database objects in the `hr` schema between the three master sites and between the master site and the materialized view site. A single replication group named `hr_repg` contains the replicated objects.

- Conflict resolution is configured for the `hr.countries` table in the multimaster environment. The latest time stamp conflict resolution method resolves conflicts on this table.
- The materialized views at the `mv1.example.com` site are updatable.

You can configure this Advanced Replication environment by completing the tasks described in the following sections of the *Oracle Database Advanced Replication Management API Reference*:

- Set up the three master sites.
- Set up the materialized view sites (to set up `mv1.example.com` only).
- Create the `hr_repg` master group for the three master sites with `orc1.example.com` as the master definition site.
- Configure time stamp conflict resolution for the `hr.countries` table.
- Create the materialized view group at `mv1.example.com` based on the `hr_repg` master group at `orc1.example.com`.

To generate the migration script for this Advanced Replication environment, complete the following steps:

1. Create the Oracle Streams Administrator at All Master Sites
2. Make a Directory Location Accessible
3. Generate the Migration Script
4. Verify the Generated Migration Script Creation and Modify Script

Step 1 Create the Oracle Streams Administrator at All Master Sites

Complete the following steps to create the Oracle Streams administrator at each master site for the replication groups being migrated to Oracle Streams. For the sample environment described in "Example Advanced Replication Environment to be Migrated to Oracle Streams" on page A-4, complete these steps at `orc1.example.com`, `orc2.example.com`, and `orc3.example.com`:

1. Connect as an administrative user who can create users, grant privileges, and create tablespaces.
2. Either create a tablespace for the Oracle Streams administrator or use an existing tablespace. For example, the following statement creates a new tablespace for the Oracle Streams administrator:

```
CREATE TABLESPACE streams_tbs DATAFILE '/usr/oracle/dbs/streams_tbs.dbf'
  SIZE 25 M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;
```

3. Create a new user to act as the Oracle Streams administrator or use an existing user. For example, to create a user named `strmadmin` and specify that this user uses the `streams_tbs` tablespace, run the following statement:

```
CREATE USER strmadmin
 IDENTIFIED BY password
   DEFAULT TABLESPACE streams_tbs
   QUOTA UNLIMITED ON streams_tbs;

GRANT DBA TO strmadmin;
```

Note:

- The migration script assumes that the user name of the Oracle Streams administrator is `strmadmin`. If your Oracle Streams administrator has a different user name, then edit the migration script to replace all instances of `strmadmin` with the user name of your Oracle Streams administrator.
- Ensure that you grant `DBA` role to the Oracle Streams administrator.

4. Grant any additional privileges required by the Oracle Streams administrator at each master site. The necessary privileges depend on your specific Oracle Streams environment.

   See Also: "Configuring an Oracle Streams Administrator on All Databases" on page 1-13 for information about addition privileges that might be required for an Oracle Streams administrator

Step 2 Make a Directory Location Accessible

The directory specified by the `file_location` parameter in the `DBMS_REPCAT.STREAMS_MIGRATION` procedure must be accessible to PL/SQL. If you do not have directory object that is accessible to the Oracle Streams administrator at the master definition site currently, then connect as the Oracle Streams administrator, and create a directory object using the SQL statement `CREATE DIRECTORY`.

A directory object is similar to an alias for the directory. For example, to create a directory object called `MIG2STR_DIR` for the `/usr/scripts` directory on your computer system, run the following procedure:

```
CONNECT strmadmin@orc1.example.com
Enter password: password

CREATE DIRECTORY MIG2STR_DIR AS '/usr/scripts';
```

See Also: *Oracle Database SQL Language Reference* for more information about the `CREATE DIRECTORY` statement

Step 3 Generate the Migration Script

To generate the migration script, run the `DBMS_REPCAT.STREAMS_MIGRATION` procedure at the master definition site and specify the appropriate parameters. For example, the following procedure generates a script that migrates an Advanced Replication environment with one replication group named `hr_repg`. The script name is `rep2streams.sql`, and it is generated into the `/usr/scripts` directory on the local computer system. This directory is represented by the directory object `MIG2STR_DIR`.

```
CONNECT strmadmin@orc1.example.com
Enter password: password

DECLARE
  rep_groups DBMS_UTILITY.NAME_ARRAY;
  BEGIN
    rep_groups(1) := 'HR_REPG';
    DBMS_REPCAT.STREAMS_MIGRATION(
      gnames         =>  rep_groups,
```

```
      file_location  =>  'MIG2STR_DIR',
      filename       =>  'rep2streams.sql');
END;
/
```

Step 4 Verify the Generated Migration Script Creation and Modify Script

After generating the migration script, verify that the script was created viewing the script in the specified directory. If necessary, you can modify it to support the following:

- If your environment requires conflict resolution that used the additive, average, priority group, or site priority Advanced Replication conflict resolution methods, then configure user-defined conflict resolution methods to resolve conflicts. Oracle Streams does not provide prebuilt conflict resolution methods that are equivalent to these methods.

  However, the migration script supports the following conflict resolution methods automatically: overwrite, discard, maximum, and minimum. The script converts an earliest time stamp method to a minimum method automatically, and it converts a latest time stamp method to a maximum method automatically. If you use a time stamp conflict resolution method, then the script assumes that any triggers necessary to populate the time stamp column in a table already exist.
- Unique conflict resolution.
- Delete conflict resolution.
- Multiple conflict resolution methods to be executed in a specified order when a conflict occurs. Oracle Streams allows only one conflict resolution method to be specified for each column list.
- Procedural replication.
- Replication of data definition language (DDL) changes for nontable objects, including the following:
  - Functions
  - Indexes
  - Indextypes
  - Operators
  - Packages
  - Package bodies
  - Procedures
  - Synonyms
  - Triggers
  - Types
  - Type bodies
  - Views

Because changes to these objects were being replicated by Advanced Replication at all sites, the migration script does not need to take any action to migrate these objects. You can add DDL rules to the Oracle Streams environment to support the future modification and creation of these types of objects.

For example, to specify that a capture process named `streams_capture` at the `orc1.example.com` database captures DDL changes to all of the database objects in the `hr` schema, add the following to the script:

```
BEGIN
  DBMS_STREAMS_ADM.ADD_SCHEMA_RULES(
    schema_name        => 'hr',
    streams_type       => 'capture',
    streams_name       => 'streams_capture',
    queue_name         => 'strmadmin.streams_queue',
    include_dml        => FALSE,
    include_ddl        => TRUE,
    include_tagged_lcr => FALSE,
    source_database    => 'orc1.example.com');
END;
/
```

Notice that the `include_ddl` parameter is set to `TRUE`. By setting this parameter to `TRUE`, this procedure adds a schema rule for DDL changes to the `hr` schema to the rule set for the capture process. This rule instructs the capture process to capture DDL changes to the `hr` schema and its objects. For the DDL changes to be replicated, you must add similar rules to the appropriate propagations and apply processes.

See Also:

- Chapter 9, "Oracle Streams Conflict Resolution"
- *Oracle Streams Concepts and Administration* for information about queue-to-queue propagations

Performing the Migration for Advanced Replication to Oracle Streams

This section explains how to perform the migration from an Advanced Replication environment to an Oracle Streams environment.

This section contains the following topics:

- Before Executing the Migration Script
- Executing the Migration Script
- After Executing the Script

Before Executing the Migration Script

Complete the following steps before executing the migration script:

1. Set Initialization Parameters That Are Relevant to Oracle Streams
2. Enable Archive Logging at All Sites
3. Create Database Links
4. Quiesce Each Replication Group That You Are Migrating to Oracle Streams

Step 1 Set Initialization Parameters That Are Relevant to Oracle Streams

At each replication database, set initialization parameters that are relevant to Oracle Streams and restart the database if necessary.

See Also: "Setting Initialization Parameters Relevant to Oracle Streams" on page 1-19 for information about initialization parameters that are important to Oracle Streams

Step 2 Enable Archive Logging at All Sites

Ensure that each master site is running in ARCHIVELOG mode, because a capture process requires ARCHIVELOG mode. In the sample environment, orc1.example.com, orc2.example.com, and orc3.example.com must be running in ARCHIVELOG mode. You can check the log mode for a database by querying the LOG_MODE column in the V$DATABASE dynamic performance view.

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

Step 3 Create Database Links

Create a database link from the Oracle Streams administrator at each master site to the Oracle Streams administrator at the other master sites. For the sample environment described in "Example Advanced Replication Environment to be Migrated to Oracle Streams" on page A-4, create the following database links:

```
CONNECT strmadmin@orc1.example.com
Enter password: password

CREATE DATABASE LINK orc2.example.com CONNECT TO strmadmin
   IDENTIFIED BY password USING 'orc2.example.com';

CREATE DATABASE LINK orc3.example.com CONNECT TO strmadmin
   IDENTIFIED BY password USING 'orc3.example.com';


CONNECT strmadmin@orc2.example.com
Enter password: password

CREATE DATABASE LINK orc1.example.com CONNECT TO strmadmin
   IDENTIFIED BY password USING 'orc1.example.com';

CREATE DATABASE LINK orc3.example.com CONNECT TO strmadmin
   IDENTIFIED BY password USING 'orc3.example.com';


CONNECT strmadmin@orc3.example.com
Enter password: password

CREATE DATABASE LINK orc1.example.com CONNECT TO strmadmin
   IDENTIFIED BY password USING 'orc1.example.com';

CREATE DATABASE LINK orc2.example.com CONNECT TO strmadmin
   IDENTIFIED BY password USING 'orc2.example.com';
```

Step 4 Quiesce Each Replication Group That You Are Migrating to Oracle Streams

Run the DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY procedure at the master definition site for each replication group that you are migrating to Oracle Streams.

In the sample environment, orc1.example.com is the master definition site, and hr_repg is the replication group being migrated to Oracle Streams. So, connect to orc1.example.com as the replication administrator and run the SUSPEND_MASTER_ACTIVITY procedure:

```
CONNECT repadmin@orc1.example.com
Enter password: password
```

```
BEGIN
   DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
      gname => 'hr_repg');
END;
/
```

Do not proceed until the master group is quiesced. You can check the status of a master group by querying the `STATUS` column in the `DBA_REPGROUP` data dictionary view.

Executing the Migration Script

Perform the following steps to migrate:

1. Connect as the Oracle Streams Administrator and Run the Script at Each Site
2. Verify That Oracle Streams Configuration Completed Successfully at All Sites

Step 1 Connect as the Oracle Streams Administrator and Run the Script at Each Site

In the sample environment, connect in SQL*Plus as the Oracle Streams administrator `strmadmin` in SQL*Plus at `orc1.example.com`, `orc2.example.com`, and `orc3.example.com` and execute the migration script `rep2streams.sql`:

```
CONNECT strmadmin@orc1.example.com
Enter password: password

SET ECHO ON
SPOOL rep2streams.out
@rep2streams.sql

CONNECT strmadmin@orc2.example.com
Enter password: password

SET ECHO ON
SPOOL rep2streams.out
@rep2streams.sql

CONNECT strmadmin@orc3.example.com
Enter password: password

SET ECHO ON
SPOOL rep2streams.out
@rep2streams.sql
```

Step 2 Verify That Oracle Streams Configuration Completed Successfully at All Sites

Check the spool file at each site to ensure that there are no errors. If there are errors, then you should modify the script to execute the steps that were not completed successfully, and then rerun the script. In the sample environment, the spool file is `rep2streams.out` at each master site.

After Executing the Script

Perform the following steps to complete the migration process:

1. Drop Replication Groups You Migrated at Each Site
2. Start the Apply Processes at Each Site

3. Start the Capture Process at Each Site

Step 1 Drop Replication Groups You Migrated at Each Site

To drop a replication group that you successfully migrated to Oracle Streams, connect as the replication administrator to the master definition site, and run the DBMS_REPCAT.DROP_MASTER_REPGROUP procedure.

> **Caution:** Ensure that the drop_contents parameter is set to FALSE in the DROP_MASTER_REPGROUP procedure. If it is set to TRUE, then the replicated database objects are dropped.

```
CONNECT repadmin@orc1.example.com
Enter password: password

BEGIN
   DBMS_REPCAT.DROP_MASTER_REPGROUP (
     gname         => 'hr_repg',
     drop_contents => FALSE,
     all_sites     => TRUE);
END;
/
```

To ensure that the migrated replication groups are dropped at each database, query the GNAME column in the DBA_REPGROUP data dictionary view. The migrated replication groups should not appear in the query output at any database.

If you no longer need the replication administrator, then you can drop this user also.

> **Caution:** Do not resume any Advanced Replication activity once Oracle Streams is set up.

Step 2 Start the Apply Processes at Each Site

You can view the names of the apply processes at each site by running the following query while connected as the Oracle Streams administrator:

```
SELECT APPLY_NAME FROM DBA_APPLY;
```

When you know the names of the apply processes, you can start each one by running the START_APPLY procedure in the DBMS_APPLY_ADM package while connected as the Oracle Streams administrator. For example, the following procedure starts an apply process named apply_from_orc2 at orc1.example.com:

```
CONNECT strmadmin@orc1.example.com
Enter password: password

BEGIN
  DBMS_APPLY_ADM.START_APPLY(
    apply_name => 'apply_from_orc2');
END;
/
```

Ensure that you start each apply process at every database in the new Oracle Streams environment.

Step 3 Start the Capture Process at Each Site

You can view the name of the capture process at each site by running the following query while connected as the Oracle Streams administrator:

```
SELECT CAPTURE_NAME FROM DBA_CAPTURE;
```

When you know the name of the capture process, you can start each one by running the `START_CAPTURE` procedure in the `DBMS_CAPTURE_ADM` package while connected as the Oracle Streams administrator. For example, the following procedure starts a capture process named `streams_capture` at `orc1.example.com`:

```
CONNECT strmadmin@orc1.example.com
Enter password: password

BEGIN
  DBMS_CAPTURE_ADM.START_CAPTURE(
    capture_name => 'streams_capture');
END;
/
```

Ensure that you start each capture process at every database in the new Oracle Streams environment.

Re-creating Master Sites to Retain Materialized View Groups

If one or more materialized view groups used a master group that you migrated to Oracle Streams, then you must re-create the master group to retain these materialized view groups. Therefore, each database acting as the master site for a materialized view group must become the master definition site for a one-master configuration of a replication group that contains the tables used by the materialized views in the materialized view group.

Use the replication management APIs to create a replication group similar to the original replication group that was migrated to Oracle Streams. That is, the new replication group should have the same replication group name, objects, conflict resolution methods, and key columns. To retain the existing materialized view groups, you must re-create each master group at each master site that contained a master group for a materialized view group, re-create the master replication objects in the master group, regenerate replication support for the master group, and resume replication activity for the master group.

For example, consider the following Advanced Replication environment:

- Two master sites, `mdb1.example.com` and `mdb2.example.com`, have the replication group `rg1`. The `mdb1.example.com` database is the master definition site, and the objects in the `rg1` replication group are replicated between `mdb1.example.com` and `mdb2.example.com`.
- The `rg1` replication group at `mdb1.example.com` is the master group to the `mvg1` materialized view group at `mv1.example.com`.
- The `rg1` replication group at `mdb2.example.com` is the master group to the `mvg2` materialized view group at `mv2.example.com`.

If the `rg1` replication group is migrated to Oracle Streams at both `mdb1.example.com` and `mdb2.example.com`, and you want to retain the materialized view groups `mvg1` at `mv1.example.com` and `mvg2` at `mv2.example.com`, then you must re-create the `rg1` replication group at `mdb1.example.com` and `mdb2.example.com` after the migration to Oracle Streams.

You configure both `mdb1.example.com` and `mdb2.example.com` to be the master definition site for the `rg1` replication group in a one-master environment.

It is not necessary to drop or re-create materialized view groups at the materialized view sites. If a new master replication group resembles the original replication group, then the materialized view groups are not affected. Do not refresh these materialized view groups until generation of replication support for each master object is complete (Step 3 in the task in this section). Similarly, do not push the deferred transaction queue at any materialized view site with updatable materialized views until generation of replication support for each master object is complete.

For the sample environment described in "Example Advanced Replication Environment to be Migrated to Oracle Streams" on page A-4, only the `hr_repg` replication group at `orc1.example.com` was the master group to a materialized view group at `mv1.example.com`. To retain this materialized view group at `mv1.example.com`, complete the following steps while connected as the replication administrator:

1. Create the master group `hr_repg` at `orc1.example.com`.

```
CONNECT repadmin@orc1.example.com
Enter password: password

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPGROUP (
      gname => 'hr_repg');
END;
/
```

2. Add the tables in the hr schema to the `hr_repg` master group. These tables are master tables to the materialized views at `mv1.example.com`.

```
BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      gname               => 'hr_repg',
      type                => 'TABLE',
      oname               => 'countries',
      sname               => 'hr',
      use_existing_object => TRUE,
      copy_rows           => FALSE);
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      gname               => 'hr_repg',
      type                => 'TABLE',
      oname               => 'departments',
      sname               => 'hr',
      use_existing_object => TRUE,
      copy_rows           => FALSE);
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      gname               => 'hr_repg',
      type                => 'TABLE',
      oname               => 'employees',
      sname               => 'hr',
```

```
      use_existing_object => TRUE,
      copy_rows           => FALSE);
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      gname               => 'hr_repg',
      type                => 'TABLE',
      oname               => 'jobs',
      sname               => 'hr',
      use_existing_object => TRUE,
      copy_rows           => FALSE);
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      gname               => 'hr_repg',
      type                => 'TABLE',
      oname               => 'job_history',
      sname               => 'hr',
      use_existing_object => TRUE,
      copy_rows           => FALSE);
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      gname               => 'hr_repg',
      type                => 'TABLE',
      oname               => 'locations',
      sname               => 'hr',
      use_existing_object => TRUE,
      copy_rows           => FALSE);
END;
/

BEGIN
   DBMS_REPCAT.CREATE_MASTER_REPOBJECT (
      gname               => 'hr_repg',
      type                => 'TABLE',
      oname               => 'regions',
      sname               => 'hr',
      use_existing_object => TRUE,
      copy_rows           => FALSE);
END;
/
```

3. Generate replication support for each object in the `hr_repg` master group.

```
BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      sname             => 'hr',
      oname             => 'countries',
      type              => 'TABLE');
END;
/
```

```
BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      sname             => 'hr',
      oname             => 'departments',
      type              => 'TABLE');
END;
/

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      sname             => 'hr',
      oname             => 'employees',
      type              => 'TABLE');
END;
/

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      sname             => 'hr',
      oname             => 'jobs',
      type              => 'TABLE');
END;
/

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      sname             => 'hr',
      oname             => 'job_history',
      type              => 'TABLE');
END;
/

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      sname             => 'hr',
      oname             => 'locations',
      type              => 'TABLE');
END;
/

BEGIN
    DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
      sname             => 'hr',
      oname             => 'regions',
      type              => 'TABLE');
END;
/
```

4. Resume master activity for the `hr_repg` master group.

```
BEGIN
   DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
      gname => 'hr_repg');
END;
/
```

Note: A materialized view log should exist for each table you added to the `hr_repg` master group, unless you deleted these logs manually after you migrated the replication group to Oracle Streams. If these materialized view logs do not exist, then you must create them.

Index

A

ABORT_GLOBAL_INSTANTIATION
procedure, 8-13
ABORT_SCHEMA_INSTANTIATION
procedure, 8-13
ABORT_SYNC_INSTANTIATION procedure, 8-13
ABORT_TABLE_INSTANTIATION procedure, 8-13
ADD SUPPLEMENTAL LOG, 1-30
ADD SUPPLEMENTAL LOG DATA, 1-31
ADD SUPPLEMENTAL LOG DATA clause of ALTER
DATABASE, 1-32
ADD SUPPLEMENTAL LOG GROUP clause of
ALTER TABLE
conditional log groups, 1-31
unconditional log groups, 1-30
alert log
Streams best practices, 15-7
ALTER DATABASE statement
ADD SUPPLEMENTAL LOG DATA clause, 1-32
DROP SUPPLEMENTAL LOG DATA
clause, 1-33
ALTER TABLE statement
ADD SUPPLEMENTAL LOG DATA clause
conditional log groups, 1-31
unconditional log groups, 1-30
ADD SUPPLEMENTAL LOG GROUP clause
conditional log groups, 1-31
unconditional log groups, 1-30
DROP SUPPLEMENTAL LOG GROUP
clause, 1-32
ALTER_APPLY procedure
removing the tag value, 10-18
setting the tag value, 10-1, 10-5, 10-18
ANYDATA data type
queues
creating, 6-1
apply process
apply handlers, 1-9
apply user
best practices, 18-1
best practices
configuration, 18-3
operation, 18-4
conflict resolution, 9-1
creating, 7-1
data types applied
heterogeneous environments, 11-4
DML changes
heterogeneous environments, 11-5
DML handlers
heterogeneous environments, 11-4
error handlers
heterogeneous, 11-4
errors
best practices, 18-4
heterogeneous environments, 11-3, 11-9
database links, 11-3
Oracle Database Gateways, 11-3
LOBs, 14-12
message handlers
heterogeneous environments, 11-4
oldest SCN
point-in-time recovery, 12-30
parallelism
best practices, 18-3
substitute key columns
heterogeneous environments, 11-4
tags, 10-5
monitoring, 10-19
removing, 10-18
setting, 10-18
update conflict handlers
monitoring, 9-17
ARCHIVELOG mode, 1-19
capture process, 5-2

B

backups
online
Streams, 10-4
Streams best practices, 15-6
batch processing
capture process best practices, 16-4
best practices
Streams replication, 14-1
alert log, 15-7
apply, 18-1
apply errors, 18-4
apply process configuration, 18-3
apply process operation, 18-4

apply process parallelism, 18-3
apply user, 18-1
archive log threads, 15-8
automate configuration, 15-2
backups, 15-6
batch processing, 16-4
capture, 16-1
capture process configuration, 16-1
capture process operation, 16-3
capture process parallelism, 16-2
capture process's queue, 15-5
capture user, 16-1
checkpoint retention, 16-2
conflict resolution, 18-2
data dictionary build, 16-4
database configuration, 15-1
database operation, 15-4
DDL replication, 1-10
destination database, 18-1
global name, 15-4
heartbeat table, 16-3
instantiation SCN, 18-2
Oracle Real Application Clusters (Oracle RAC) databases, 15-8
performance, 15-5
prepare for instantiation, 16-4
propagation, 17-1
propagation latency, 17-2
queue-to-queue propagation, 17-1
removing, 15-8
restarting propagation, 17-3
SDU, 17-3
statistics collection, 15-7
synchronous capture configuration, 16-4
bi-directional replication, 1-7, 1-12

C

capture process
ARCHIVELOG mode, 5-2
best practices
batch processing, 16-4
configuration, 16-1
data dictionary build, 16-4
operation, 16-3
prepare for instantiation, 16-4
capture user
best practices, 16-1
configuring, 5-1
creating, 5-1
DBID, 5-2
changing, 12-23
downstream capture, 1-5, 2-7
creating, 5-1
global name, 5-2
changing, 12-23
heterogeneous environments, 11-2
local capture, 1-5, 2-7
log sequence number
resetting, 12-25
parallelism
best practices, 16-2
parameters
merge_threshold, 12-12
message_tracking_frequency, 12-1
split_threshold, 12-12
preparing for, 5-2
supplemental logging, 1-28, 8-6
change cycling
avoidance
tags, 10-6
checkpoint retention
best practices, 16-2
column lists, 9-9
COMPARE function, 13-7
perform_row_dif parameter, 13-9
COMPARE_OLD_VALUES procedure, 9-4, 9-14
comparing database objects, 13-7
COMPATIBLE initialization parameter, 1-20
conflict resolution, 9-1
best practices, 18-2
column lists, 9-9
conflict handlers, 9-3, 9-4, 9-6
custom, 9-11
modifying, 9-13
prebuilt, 9-6
removing, 9-14
setting, 9-12
data convergence, 9-11
DISCARD handler, 9-7
MAXIMUM handler, 9-7
latest time, 9-8
MINIMUM handler, 9-8
OVERWRITE handler, 9-7
resolution columns, 9-10
time-based, 9-8
conflicts
avoidance, 9-4
delete, 9-5
primary database ownership, 9-4
sequences, 9-5
uniqueness, 9-5
update, 9-5
delete, 9-2
detection, 9-3
identifying rows, 9-4
monitoring, 9-16
stopping, 9-4, 9-14
DML conflicts, 9-1
foreign key, 9-2
transaction ordering, 9-3
types of, 9-2
uniqueness, 9-2
update, 9-2
CONVERGE procedure, 13-30
CREATE_APPLY procedure, 7-1
tags, 10-1, 10-5
CREATE_CAPTURE procedure, 5-1, 5-4
CREATE_COMPARISON procedure, 13-7
CREATE_PROPAGATION procedure, 6-3

CREATE_SYNC_CAPTURE procedure, 5-20

D

data
comparing, 13-7
custom, 13-14
cyclic, 13-12
purging results, 13-35
random, 13-11
rechecking, 13-34
subset of columns, 13-8
converging, 13-30
session tags, 13-33
data types
heterogeneous environments, 11-4
database links, 1-17
databases
adding for replication, 4-7, 4-15, 4-25
DBA_APPLY view, 10-19
DBA_APPLY_CONFLICT_COLUMNS view, 9-17
DBA_APPLY_INSTANTIATED_OBJECTS view, 8-42
DBA_APPLY_TABLE_COLUMNS view, 9-16
DBA_CAPTURE_PREPARED_DATABASE view, 8-41
DBA_CAPTURE_PREPARED_SCHEMAS view, 8-41
DBA_CAPTURE_PREPARED_TABLES view, 8-41
DBA_COMPARISON view, 13-19, 13-26, 13-27
DBA_COMPARISON_COLUMNS view, 13-22
DBA_COMPARISON_SCAN view, 13-24, 13-26, 13-27
DBA_COMPARISON_SCAN_VALUES view, 13-28
DBA_RECOVERABLE_SCRIPT view, 12-36
DBA_RECOVERABLE_SCRIPT_BLOCKS view, 12-36
DBA_RECOVERABLE_SCRIPT_ERRORS view, 12-36
DBA_RECOVERABLE_SCRIPT_PARAMS view, 12-36
DBA_SYNC_CAPTURE_PREPARED_TABS view, 8-41
DBID (database identifier)
capture process, 5-2
changing, 12-23
DBMS_CAPTURE_ADM package, 5-1
DBMS_COMPARISON package
buckets, 13-2
comparing database objects, 13-7
custom, 13-14
cyclic, 13-12
purging results, 13-35
random, 13-11
rechecking, 13-34
subset of columns, 13-8
converging database objects, 13-30
session tags, 13-33
monitoring, 13-19
parent scans, 13-3
preparation, 13-6
root scans, 13-3
scans, 13-2
Streams replication, 13-37
using, 13-1
DBMS_PROPAGATION_ADM package, 6-1
DBMS_STREAMS package, 10-16
DBMS_STREAMS_ADM package, 1-3, 5-1, 6-1
creating a capture process, 5-1
creating a propagation, 6-3
creating an apply process, 7-1
preparation for instantiation, 8-4
tags, 10-2
DDL replication
best practices, 1-10
directory objects, 2-15
creating, 4-9
DISCARD conflict resolution handler, 9-7
DML handlers
LOB assembly, 14-12
downstream capture, 1-5, 2-7
archived-log, 1-6
configuring, 2-32
log file transfer, 1-33
real-time, 1-6, 1-36
standby redo logs, 1-36
DROP SUPPLEMENTAL LOG DATA clause of ALTER DATABASE, 1-33
DROP SUPPLEMENTAL LOG GROUP clause, 1-32
DROP_COMPARISON procedure, 13-36

E

ENQUEUE procedure, 14-4
error handlers
LOB assembly, 14-12
error queue
heterogeneous environments, 11-8
Export
Oracle Streams, 8-37

F

flashback queries
Streams replication, 12-34

G

GET_MESSAGE_TRACKING function, 12-1
GET_SCN_MAPPING procedure, 12-29, 12-34
GET_TAG function, 10-17, 10-18
global name
best practices, 15-4
capture process, 5-2
changing, 12-23
GLOBAL_NAMES initialization parameter, 1-20
GRANT_REMOTE_ADMIN_ACCESS procedure, 5-8, 5-11

H

heartbeat table
  Streams best practices, 16-3
heterogeneous information sharing, 11-1
  non-Oracle to non-Oracle, 11-10
  non-Oracle to Oracle, 11-8
    apply process, 11-9
    capturing changes, 11-9
    instantiation, 11-10
    user application, 11-9
  Oracle to non-Oracle, 11-1
    apply process, 11-3
    capture process, 11-2
    data types applied, 11-4
    database links, 11-3
    DML changes, 11-5
    DML handlers, 11-4
    error handlers, 11-4
    errors, 11-8
    instantiation, 11-5
    message handlers, 11-4
    staging, 11-2
    substitute key columns, 11-4
    transformations, 11-7
hub-and-spoke replication, 1-7, 1-12, 10-9
  configuring, 2-48

I

Import
  Oracle Streams, 8-37
  STREAMS_CONFIGURATION parameter, 8-15
initialization parameters
  COMPATIBLE, 1-20
  GLOBAL_NAMES, 1-20
  LOG_ARCHIVE_CONFIG, 1-20
  LOG_ARCHIVE_DEST_*n*, 1-20
  LOG_ARCHIVE_DEST_STATE_*n*, 1-21
  LOG_BUFFER, 1-21
  MEMORY_MAX_TARGET, 1-21
  MEMORY_TARGET, 1-21
  OPEN_LINKS, 1-21
  PROCESSES, 1-22
  SESSIONS, 1-22
  SGA_MAX_SIZE, 1-22
  SGA_TARGET, 1-22
  SHARED_POOL_SIZE, 1-23
  STREAMS_POOL_SIZE, 1-24
  TIMED_STATISTICS, 1-25
  UNDO_RETENTION, 1-25
instantiation, 2-13, 8-1
  aborting preparation, 8-13
  Data Pump, 8-13
  database, 8-25
  example
    Data Pump export/import, 8-17
    RMAN CONVERT DATABASE, 8-31
    RMAN DUPLICATE, 8-26
    RMAN TRANSPORT TABLESPACE, 8-20
    transportable tablespace, 8-20
  heterogeneous environments
    non-Oracle to Oracle, 11-10
    Oracle to non-Oracle, 11-5
  monitoring, 8-41
  Oracle Streams, 8-37
  preparation for, 8-3
  preparing for, 8-1, 8-8
  RMAN, 8-19
  setting an SCN, 8-36
    DDL LCRs, 8-38
    export/import, 8-37
  supplemental logging specifications, 8-2
instantiation SCN
  best practices, 18-2

L

LCRs. *See* logical change records
LOB assembly, 14-12
LOBs
  Oracle Streams, 14-11
    apply process, 14-12
log sequence number
  Streams capture process, 12-25
LOG_ARCHIVE_CONFIG initialization parameter, 1-20
LOG_ARCHIVE_DEST_*n* initialization parameter, 1-20
LOG_ARCHIVE_DEST_STATE_*n* initialization parameter, 1-21
LOG_BUFFER initialization parameter, 1-21
logical change records (LCRs)
  constructing, 14-2
  enqueuing, 14-2
  executing, 14-6
    DDL LCRs, 14-10
    row LCRs, 14-7
  LOB columns, 14-11, 14-21
    apply process, 14-12
    requirements, 14-18
  LONG columns, 14-21
    requirements, 14-21
  LONG RAW columns, 14-21
    requirements, 14-21
  managing, 14-1
  requirements, 14-1
  tracking, 12-1
  XMLType, 14-11
LONG data type
  Oracle Streams, 14-21
LONG RAW data type
  Oracle, 14-21

M

MAINTAIN_GLOBAL procedure, 2-3, 2-16
MAINTAIN_SCHEMAS procedure, 2-3, 2-24, 2-38, 2-42, 2-48
MAINTAIN_SIMPLE_TTS procedure, 2-3, 2-33
MAINTAIN_TABLES procedure, 2-3, 2-27

MAINTAIN_TTS procedure, 2-3, 2-33
MAXIMUM conflict resolution handler, 9-7
  latest time, 9-8
MEMORY_MAX_TARGET initialization parameter, 1-21, 1-26
MEMORY_TARGET initialization parameter, 1-21, 1-26
merge streams, 12-5
MERGE_STREAMS procedure, 12-13
MERGE_STREAMS_JOB procedure, 12-13
message tracking, 12-1
message_tracking_frequency capture process parameter, 12-1
MINIMUM conflict resolution handler, 9-8
monitoring
  apply process
    update conflict handlers, 9-17
  comparisons, 13-19
  conflict detection, 9-16
  instantiation, 8-41
  tags, 10-18
    apply process value, 10-19
    current session value, 10-18

N

n-way replication, 1-7, 1-13, 10-6

O

objects
  adding for replication, 4-4, 4-11, 4-18
oldest SCN
  point-in-time recovery, 12-30
one-way replication, 1-12
OPEN_LINKS initialization parameter, 1-21
optimizer
  statistics collection
    best practices, 15-7
Oracle Data Pump
  Import utility
    STREAMS_CONFIGURATION parameter, 8-15
  instantiations, 8-17
  Streams instantiation, 8-13
Oracle Database Gateways
  Oracle Streams, 11-3
Oracle Real Application Clusters
  Streams best practices, 15-8
    archive log threads, 15-8
    global name, 15-8
    propagations, 15-9
    queue ownership, 15-9
Oracle Streams
  conflict resolution, 9-1
  DBMS_COMPARISON package, 13-37
  Export utility, 8-37
  heterogeneous information sharing, 11-1
  Import utility, 8-37
  instantiation, 8-1, 8-37
  LOBs, 14-11
  logical change records (LCRs)
    managing, 14-1
  LONG data type, 14-21
  migrating to from Advanced Replication, A-1
  Oracle Database Gateways, 11-3
  point-in-time recovery, 12-25
  replication, 1-1
    adding databases, 4-7, 4-15, 4-25
    adding objects, 4-4, 4-11, 4-18
    adding to, 4-1
    best practices, 14-1
    configuring, 2-1, 3-1
    managing, 12-1
    sequences, 9-5
  rules, 1-3
  sequences, 9-5
  tags, 10-1
  XMLType, 14-11
Oracle Streams Performance Advisor, 15-5
OVERWRITE conflict resolution handler, 9-7

P

performance
  Oracle Streams, 15-5
point-in-time recovery
  Oracle Streams, 12-25
POST_INSTANTIATION_SETUP procedure, 2-3, 2-16, 2-33
PRE_INSTANTIATION_SETUP procedure, 2-3, 2-16, 2-33
PREPARE_GLOBAL_INSTANTIATION procedure, 8-3, 8-8
PREPARE_SCHEMA_INSTANTIATION procedure, 8-3, 8-8
PREPARE_SYNC_INSTANTIATION function, 8-3, 8-8
PREPARE_TABLE_INSTANTIATION procedure, 8-3, 8-8
PROCESSES initialization parameter, 1-22
propagation
  best practices, 17-1
    broken propagations, 17-3
    configuration, 17-1
    propagation latency, 17-2
    propagation operation, 17-3
    queue-to-queue propagation, 17-1
    restarting propagation, 17-3
    SDU, 17-3
propagation jobs
  managing, 6-3
propagations
  creating, 6-1, 6-3
  managing, 6-3
PURGE_COMPARISON procedure, 13-35

Q

queues

ANYDATA
creating, 6-1
commit-time, 11-9
creating, 6-1
size
best practices, 15-5
transactional, 11-9
queue-to-queue propagation
best practices, 17-1

R

RECHECK function, 13-34
RECOVER_OPERATION procedure, 12-36
Recovery Manager
CONVERT DATABASE command
Streams instantiation, 8-31
DUPLICATE command
Streams instantiation, 8-26
Streams instantiation, 8-19
TRANSPORT TABLESPACE command
Streams instantiation, 8-20
replication
adding databases, 4-7, 4-15, 4-25
adding objects, 4-4, 4-11, 4-18
adding to, 4-1
bi-directional, 1-4, 2-10
configuration errors
recovering, 12-36
configuring, 2-1, 3-1
apply handlers, 1-9
ARCHIVELOG mode, 1-19
bi-directional, 1-7, 1-12
database, 2-16
database links, 1-17
DBMS_STREAMS_ADM package, 2-3
DDL changes, 1-10, 2-13
directory objects, 2-15
downstream capture, 1-5, 2-7, 2-32
Enterprise Manager, 2-1
hub-and-spoke, 1-7, 1-12, 2-48
initialization parameters, 1-19
instantiation, 2-13
local capture, 1-5, 2-7
log file transfer, 1-33
multiple-source environment, 3-6
n-way, 1-7, 1-13
one-way, 1-12
Oracle Streams pool, 1-25
preparation, 1-1
schemas, 2-24, 2-38, 2-42, 2-48
scripts, 2-8
single-source environment, 3-2
standby redo logs, 1-36
supplemental logging, 1-28
tables, 2-27
tablespace, 2-33
tablespaces, 2-33
tags, 2-11
hub-and-spoke, 1-4, 10-9
managing, 12-1
migrating to Streams, A-1
n-way, 1-4, 10-6
one-way, 1-4, 2-10
Oracle Streams, 1-1
best practices, 14-1
managing, 12-1
sequences, 9-5
split and merge, 12-5
resolution columns, 9-10
rule-based transformations, 1-8
rules, 1-3
system-created
tags, 10-2

S

SDU
Streams best practices, 17-3
sequences, 9-5
replication, 9-5
SESSIONS initialization parameter, 1-22
SET_DML_HANDLER procedure, 9-11
SET_GLOBAL_INSTANTIATION_SCN
procedure, 8-36, 8-38
SET_MESSAGE_TRACKING procedure, 12-1
SET_SCHEMA_INSTANTIATION_SCN
procedure, 8-36, 8-38
SET_TABLE_INSTANTIATION_SCN
procedure, 8-36
SET_TAG procedure, 10-1, 10-17
SET_UP_QUEUE procedure, 6-1
SET_UPDATE_CONFLICT_HANDLER
procedure, 9-6
modifying an update conflict handler, 9-13
removing an update conflict handler, 9-14
setting an update conflict handler, 9-12
SGA_MAX_SIZE initialization parameter, 1-22
SGA_TARGET initialization parameter, 1-22, 1-27
SHARED_POOL_SIZE initialization parameter, 1-23
split streams, 12-5
SPLIT_STREAMS procedure, 12-13
staging
heterogeneous environments, 11-2
statistics
Oracle Streams, 15-5
Streams pool
MEMORY_MAX_TARGET initialization
parameter, 1-26
MEMORY_TARGET initialization
parameter, 1-26
SGA_TARGET initialization parameter, 1-27
STREAMS_CONFIGURATION parameter
Data Pump Import utility, 8-15
Import utility, 8-15
STREAMS_MIGRATION procedure, A-1
STREAMS_POOL_SIZE initialization
parameter, 1-24
STRMMON, 15-5
supplemental logging, 1-28

column lists, 9-9
instantiation, 8-2
preparation for instantiation, 8-6, 8-8
synchronous capture
best practices
configuration, 16-4
configuring, 5-17
preparing for, 5-18
system change numbers (SCN)
oldest SCN for an apply process
point-in-time recovery, 12-30

T

tags, 2-11, 10-1
ALTER_APPLY procedure, 10-1, 10-5
apply process, 10-5
change cycling
avoidance, 10-6
CONVERGE procedure, 13-33
CREATE_APPLY procedure, 10-1, 10-5
examples, 10-6
getting value for current session, 10-17
hub-and-spoke replication, 10-6
managing, 10-16
monitoring, 10-18
apply process value, 10-19
current session value, 10-18
n-way replication, 10-6
online backups, 10-4
removing value for apply process, 10-18
rules, 10-2
include_tagged_lcr parameter, 10-3
SET_TAG procedure, 10-1
setting value for apply process, 10-18
setting value for current session, 10-17
TIMED_STATISTICS initialization parameter, 1-25
tracking LCRs, 12-1
transformations
heterogeneous environments
Oracle to non-Oracle, 11-7
rule-based, 1-8
transportable tablespace
Streams instantiation, 8-20

U

UNDO_RETENTION initialization parameter, 1-25

V

V$STREAMS_MESSAGE_TRACKING view, 12-1
V$STREAMS_POOL_ADVICE view, 1-27
V$STREAMS_TRANSACTION view, 8-8

X

XMLType
logical change records (LCRs), 14-11