

Oracle® Data Guard
Concepts and Administration
10g Release 2 (10.2)
B14239-04

March 2006

Oracle Data Guard Concepts and Administration, 10g Release 2 (10.2)

B14239-04

Copyright © 1999, 2006, Oracle. All rights reserved.

Primary Author: Vivian Schupmann

Contributors: Andy Adams, Beldalker Anand, Rick Anderson, Andrew Babb, Pam Bantis, Tammy Bednar, Barbara Benton, Chipper Brown, Larry Carpenter, George Claborn, Laurence Clarke, Jay Davison, Jeff Detjen, Ray Dutcher, B.G. Garin, Mahesh Girkar, Yosuke Goto, Ray Guzman, Susan Hillson, Mark Johnson, Rajeev Jain, Joydip Kundu, J. William Lee, Steve Lee, Steve Lim, Nitin Karkhanis, Steve McGee, Bob McGuirk, Joe Meeks, Steve Moriarty, Muthu Olagappan, Deborah Owens, Ashish Ray, Antonio Romero, Mike Schloss, Mike Smith, Vinay Srihali, Morris Tao, Lawrence To, Doug Utzig, Ric Van Dyke, Doug Voss, Ron Weiss, Jingming Zhang

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

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

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States 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, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software—Restricted Rights (June 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065

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

Oracle, JD Edwards, PeopleSoft, and Retek are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

Contents

Preface	xvii
Audience	xvii
Documentation Accessibility	xvii
Related Documents	xviii
Conventions	xviii
What's New in Oracle Data Guard?	xix
Part I Concepts and Administration	
1 Introduction to Oracle Data Guard	
1.1 Data Guard Configurations	1-1
1.1.1 Primary Database	1-2
1.1.2 Standby Databases	1-2
1.1.3 Configuration Example	1-2
1.2 Data Guard Services	1-3
1.2.1 Redo Transport Services	1-3
1.2.2 Log Apply Services	1-4
1.2.3 Role Transitions	1-5
1.3 Data Guard Broker	1-5
1.3.1 Using Oracle Enterprise Manager	1-6
1.3.2 Using the Data Guard Command-Line Interface	1-7
1.4 Data Guard Protection Modes	1-7
1.5 Data Guard and Complementary Technologies	1-8
1.6 Summary of Data Guard Benefits	1-9
2 Getting Started with Data Guard	
2.1 Standby Database Types	2-1
2.1.1 Physical Standby Databases	2-1
2.1.2 Logical Standby Databases	2-3
2.2 User Interfaces for Administering Data Guard Configurations	2-4
2.3 Data Guard Operational Prerequisites	2-4
2.3.1 Hardware and Operating System Requirements	2-4
2.3.2 Oracle Software Requirements	2-5
2.4 Standby Database Directory Structure Considerations	2-6

2.5	Online Redo Logs, Archived Redo Logs, and Standby Redo Logs	2-8
2.5.1	Online Redo Logs and Archived Redo Logs	2-8
2.5.2	Standby Redo Logs.....	2-9

3 Creating a Physical Standby Database

3.1	Preparing the Primary Database for Standby Database Creation	3-1
3.1.1	Enable Forced Logging	3-2
3.1.2	Create a Password File.....	3-2
3.1.3	Configure a Standby Redo Log.....	3-2
3.1.4	Set Primary Database Initialization Parameters.....	3-4
3.1.5	Enable Archiving	3-7
3.2	Step-by-Step Instructions for Creating a Physical Standby Database.....	3-7
3.2.1	Create a Backup Copy of the Primary Database Datafiles	3-7
3.2.2	Create a Control File for the Standby Database	3-8
3.2.3	Prepare an Initialization Parameter File for the Standby Database	3-8
3.2.4	Copy Files from the Primary System to the Standby System.....	3-10
3.2.5	Set Up the Environment to Support the Standby Database	3-10
3.2.6	Start the Physical Standby Database.....	3-11
3.2.7	Verify the Physical Standby Database Is Performing Properly	3-12
3.3	Post-Creation Steps.....	3-13

4 Creating a Logical Standby Database

4.1	Prerequisite Conditions for Creating a Logical Standby Database	4-1
4.1.1	Determine Support for Data Types and Storage Attributes for Tables	4-1
4.1.2	Ensure Table Rows in the Primary Database Can Be Uniquely Identified	4-1
4.2	Step-by-Step Instructions for Creating a Logical Standby Database.....	4-3
4.2.1	Create a Physical Standby Database	4-3
4.2.2	Stop Redo Apply on the Physical Standby Database	4-3
4.2.3	Prepare the Primary Database to Support a Logical Standby Database.....	4-3
4.2.3.1	Prepare the Primary Database for Role Transitions	4-4
4.2.3.2	Build a Dictionary in the Redo Data.....	4-4
4.2.4	Transition to a Logical Standby Database.....	4-5
4.2.4.1	Convert to a Logical Standby Database	4-5
4.2.4.2	Create a New Password File	4-5
4.2.4.3	Adjust Initialization Parameters for the Logical Standby Database	4-5
4.2.5	Open the Logical Standby Database	4-7
4.2.6	Verify the Logical Standby Database Is Performing Properly	4-7
4.3	Post-Creation Steps.....	4-7

5 Redo Transport Services

5.1	Introduction to Redo Transport Services	5-1
5.2	Where to Send Redo Data.....	5-2
5.2.1	Destination Types	5-2
5.2.2	Configuring Destinations with the LOG_ARCHIVE_DEST_n Parameter.....	5-3
5.2.2.1	Changing Destination Attributes	5-5
5.2.2.2	Viewing Attribute with V\$ARCHIVE_DEST	5-5

5.2.3	Setting Up Flash Recovery Areas	5-5
5.2.3.1	Using the LOG_ARCHIVE_DEST_10 Destination	5-6
5.2.3.2	Using Other LOG_ARCHIVE_DEST_n Destinations	5-6
5.2.3.3	Using the STANDBY_ARCHIVE_DEST Destination.....	5-7
5.2.3.4	Sharing a Flash Recovery Area Between Primary and Standby Databases.....	5-7
5.3	How to Send Redo Data.....	5-7
5.3.1	Using Archiver Processes (ARCn) to Archive Redo Data	5-8
5.3.1.1	Initialization Parameters That Control ARCn Archival Behavior.....	5-8
5.3.1.2	ARCn Archival Processing.....	5-8
5.3.2	Using the Log Writer Process (LGWR) to Archive Redo Data	5-10
5.3.2.1	LOG_ARCHIVE_DEST_n Attributes for LGWR Archival Processing.....	5-10
5.3.2.2	LGWR SYNC Archival Processing.....	5-11
5.3.2.3	LGWR ASYNC Archival Processing	5-12
5.3.3	Providing for Secure Redo Data Transmission	5-13
5.4	When Redo Data Should Be Sent.....	5-14
5.4.1	Specifying Role-Based Destinations with the VALID_FOR Attribute.....	5-14
5.4.2	Specify Unique Names for Primary and Standby Databases.....	5-15
5.5	What to Do If Errors Occur.....	5-16
5.5.1	Retrying the Archival Operation.....	5-16
5.5.2	Using an Alternate Destination	5-17
5.5.3	Controlling the Number of Retry Attempts	5-17
5.6	Setting Up a Data Protection Mode.....	5-18
5.6.1	Choosing a Data Protection Mode	5-18
5.6.1.1	Maximum Protection Mode	5-18
5.6.1.2	Maximum Availability Mode	5-18
5.6.1.3	Maximum Performance Mode.....	5-19
5.6.2	Setting the Data Protection Mode of a Data Guard Configuration	5-19
5.7	Managing Log Files	5-21
5.7.1	Specifying Alternate Directory Locations for Archived Redo Log Files	5-21
5.7.2	Reusing Online Redo Log Files.....	5-22
5.7.3	Managing Standby Redo Log Files	5-23
5.7.3.1	Determining If a Standby Redo Log File Group Configuration Is Adequate...	5-23
5.7.3.2	Adding Standby Redo Log Members to an Existing Group	5-23
5.7.4	Planning for Growth and Reuse of the Control Files	5-23
5.7.4.1	Sizing the Disk Volumes that Contain the Control Files	5-24
5.7.4.2	Specifying the Reuse of Records in the Control File	5-24
5.7.5	Sharing a Log File Destination Among Multiple Standby Databases.....	5-24
5.8	Managing Archive Gaps	5-25
5.8.1	When Is an Archive Gap Discovered?.....	5-26
5.8.2	How Is a Gap Resolved?	5-26
5.8.3	Using the Fetch Archive Log (FAL) to Resolve Archive Gaps.....	5-26
5.8.4	Manually Determining and Resolving Archive Gaps.....	5-27
5.9	Verification.....	5-29
5.9.1	Monitoring Log File Archival Information.....	5-29
5.9.2	Monitoring the Performance of Redo Transport Services	5-30
5.9.2.1	ARCn Process Wait Events	5-30
5.9.2.2	LGWR SYNC Wait Events.....	5-30

5.9.2.3	LGWR ASYNC Wait Events	5-31
---------	------------------------------	------

6 Log Apply Services

6.1	Introduction to Log Apply Services	6-1
6.2	Log Apply Services Configuration Options.....	6-2
6.2.1	Using Real-Time Apply to Apply Redo Data Immediately	6-2
6.2.2	Specifying a Time Delay for the Application of Archived Redo Log Files	6-3
6.2.2.1	Using Flashback Database as an Alternative to Setting a Time Delay	6-4
6.3	Applying Redo Data to Physical Standby Databases	6-4
6.3.1	Starting Redo Apply	6-4
6.3.2	Stopping Redo Apply.....	6-5
6.3.3	Monitoring Redo Apply on Physical Standby Databases.....	6-5
6.4	Applying Redo Data to Logical Standby Databases.....	6-5
6.4.1	Starting SQL Apply	6-5
6.4.2	Stopping SQL Apply on a Logical Standby Database.....	6-5
6.4.3	Monitoring SQL Apply on Logical Standby Databases	6-6

7 Role Transitions

7.1	Introduction to Role Transitions.....	7-1
7.1.1	Preparing for a Role Transition (Failover or Switchover).....	7-2
7.1.2	Choosing a Target Standby Database for a Role Transition.....	7-3
7.1.3	Switchovers.....	7-4
7.1.4	Failovers	7-6
7.2	Role Transitions Involving Physical Standby Databases	7-7
7.2.1	Switchovers Involving a Physical Standby Database.....	7-7
7.2.2	Failovers Involving a Physical Standby Database	7-9
7.3	Role Transitions Involving Logical Standby Databases.....	7-13
7.3.1	Switchovers Involving a Logical Standby Database.....	7-13
7.3.2	Failovers Involving a Logical Standby Database	7-16
7.4	Using Flashback Database After a Role Transition.....	7-20
7.4.1	Using Flashback Database After a Switchover.....	7-20
7.4.2	Using Flashback Database After a Failover	7-20

8 Managing a Physical Standby Database

8.1	Starting Up and Shutting Down a Physical Standby Database	8-1
8.1.1	Starting Up a Physical Standby Database	8-1
8.1.2	Shutting Down a Physical Standby Database.....	8-2
8.2	Opening a Standby Database for Read-Only or Read/Write Access	8-2
8.2.1	Assessing Whether or Not to Open a Standby Database.....	8-3
8.2.2	Opening a Physical Standby Database for Read-Only Access.....	8-4
8.3	Managing Primary Database Events That Affect the Standby Database.....	8-5
8.3.1	Adding a Datafile or Creating a Tablespace	8-6
8.3.1.1	When STANDBY_FILE_MANAGEMENT Is Set to AUTO	8-6
8.3.1.2	When STANDBY_FILE_MANAGEMENT Is Set to MANUAL	8-7
8.3.2	Dropping Tablespaces and Deleting Datafiles	8-9
8.3.2.1	When STANDBY_FILE_MANAGEMENT Is Set to AUTO or MANUAL.....	8-9

8.3.2.2	Using DROP TABLESPACE INCLUDING CONTENTS AND DATAFILES...	8-10
8.3.3	Using Transportable Tablespaces with a Physical Standby Database	8-10
8.3.4	Renaming a Datafile in the Primary Database	8-10
8.3.5	Adding or Dropping Online Redo Log Files	8-12
8.3.6	NOLOGGING or Unrecoverable Operations	8-12
8.4	Recovering Through the OPEN RESETLOGS Statement.....	8-13
8.5	Monitoring the Primary and Standby Databases	8-13
8.5.1	Alert Log	8-15
8.5.2	Dynamic Performance Views (Fixed Views)	8-15
8.5.3	Monitoring Recovery Progress	8-15
8.5.3.1	Monitoring the Process Activities	8-15
8.5.3.2	Determining the Progress of Redo Apply.....	8-16
8.5.3.3	Determining the Location and Creator of the Archived Redo Log Files.....	8-16
8.5.3.4	Viewing Database Incarnations Before and After OPEN RESETLOGS	8-17
8.5.3.5	Viewing the Archived Redo Log History	8-17
8.5.3.6	Determining Which Log Files Were Applied to the Standby Database	8-18
8.5.3.7	Determining Which Log Files Were Not Received by the Standby Site.....	8-18
8.5.4	Monitoring Log Apply Services on Physical Standby Databases.....	8-19
8.5.4.1	Accessing the V\$DATABASE View	8-19
8.5.4.2	Accessing the V\$MANAGED_STANDBY Fixed View	8-19
8.5.4.3	Accessing the V\$ARCHIVE_DEST_STATUS Fixed View	8-20
8.5.4.4	Accessing the V\$ARCHIVED_LOG Fixed View	8-20
8.5.4.5	Accessing the V\$LOG_HISTORY Fixed View	8-20
8.5.4.6	Accessing the V\$DATAGUARD_STATUS Fixed View.....	8-21
8.6	Tuning the Log Apply Rate for a Physical Standby Database	8-22

9 Managing a Logical Standby Database

9.1	Overview of the SQL Apply Architecture.....	9-1
9.1.1	Various Considerations for SQL Apply	9-2
9.1.1.1	Transaction Size Considerations	9-2
9.1.1.2	Pageout Considerations.....	9-3
9.1.1.3	Restart Considerations.....	9-3
9.1.1.4	DML Apply Considerations.....	9-4
9.1.1.5	DDL Apply Considerations	9-4
9.2	Views Related to Managing and Monitoring a Logical Standby Database.....	9-4
9.2.1	DBA_LOGSTDBY_EVENTS View	9-5
9.2.2	DBA_LOGSTDBY_LOG View	9-5
9.2.3	V\$LOGSTDBY_STATS View	9-6
9.2.4	V\$LOGSTDBY_PROCESS View	9-6
9.2.5	V\$LOGSTDBY_PROGRESS View	9-8
9.2.6	V\$LOGSTDBY_STATE View	9-9
9.2.7	V\$LOGSTDBY_STATS View	9-9
9.3	Monitoring a Logical Standby Database	9-10
9.3.1	Monitoring SQL Apply Progress.....	9-10
9.3.2	Automatic Deletion of Log Files.....	9-12
9.4	Customizing a Logical Standby Database.....	9-13
9.4.1	Using Real-Time Apply On the Logical Standby Database	9-14

9.4.2	Customizing Logging of Events in the DBA_LOGSTDBY_EVENTS View	9-14
9.4.3	Using DBMS_LOGSTDBY.SKIP to Prevent Changes to Specific Schema Objects..	9-15
9.4.4	Setting up a Skip Handler for a DDL Statement	9-15
9.4.5	Modifying a Logical Standby Database.....	9-16
9.4.5.1	Performing DDL on a Logical Standby Database.....	9-16
9.4.5.2	Modifying Tables That Are Not Maintained by SQL Apply	9-17
9.4.6	Adding or Re-Creating Tables On a Logical Standby Database.....	9-18
9.5	Managing Specific Workloads In the Context of a Logical Standby Database.....	9-19
9.5.1	Importing a Transportable Tablespace to the Primary Database	9-19
9.5.2	Using Materialized Views	9-20
9.5.3	How Triggers and Constraints Are Handled on a Logical Standby Database.....	9-20
9.5.4	Recovering Through the OPEN RESETLOGS Statement	9-20
9.6	Tuning a Logical Standby Database.....	9-21
9.6.1	Create a Primary Key RELY Constraint	9-21
9.6.2	Gather Statistics for the Cost-Based Optimizer.....	9-22
9.6.3	Adjust the Number of Processes	9-23
9.6.3.1	Adjusting the Number of APPLIER Processes.....	9-23
9.6.3.2	Adjusting the Number of PREPARER Processes.....	9-24
9.6.4	Adjust the Memory Used for LCR Cache.....	9-25
9.6.5	Adjust How Transactions are Applied On the Logical Standby Database	9-26

10 Using RMAN to Back Up and Restore Files

10.1	Backup Procedure.....	10-1
10.1.1	Using Disk as Cache for Tape Backup	10-1
10.1.2	Performing Backups Directly to Tape.....	10-2
10.2	Effect of Switchovers, Failovers, and Control File Creation on Backups	10-2
10.2.1	Recovery from Loss of Datafiles on the Primary Database	10-3
10.2.2	Recovery from Loss of Datafiles on the Standby Database	10-3
10.2.3	Recovery from the Loss of a Standby Control File	10-4
10.2.4	Recovery from the Loss of the Primary Control File.....	10-4
10.2.5	Recovery from the Loss of an Online Redo Log File	10-5
10.2.6	Incomplete Recovery of the Database.....	10-5
10.3	Additional Backup Situations	10-7
10.3.1	Standby Databases Too Geographically Distant to Share Backups	10-7
10.3.2	Standby Database Does Not Contain Datafiles, Used as a FAL Server.....	10-7
10.3.3	Standby Database File Names Are Different than Primary Database	10-8
10.3.4	Deletion Policy for Archived Redo Log Files In Flash Recovery Areas	10-8
10.3.4.1	Reconfiguring the Deletion Policy After a Role Transition.....	10-9
10.3.4.2	Viewing the Current Deletion Policy	10-9

11 Using SQL Apply to Upgrade the Oracle Database

11.1	Benefits of a Rolling Upgrade Using SQL Apply.....	11-1
11.2	Requirements to Perform a Rolling Upgrade Using SQL Apply.....	11-2
11.3	Figures and Conventions Used in the Upgrade Instructions.....	11-2
11.4	Prepare to Upgrade	11-3
11.5	Upgrade the Databases	11-5

12 Data Guard Scenarios

12.1	Setting Up and Verifying Archival Destinations	12-1
12.1.1	Configuring a Primary Database and a Physical Standby Database	12-1
12.1.2	Configuring a Primary Database and a Logical Standby Database	12-3
12.1.3	Configuring Both Physical and Logical Standby Databases	12-6
12.1.4	Verifying the Current VALID_FOR Attribute Settings for Each Destination.....	12-9
12.2	Choosing the Best Available Standby Database for a Role Transition.....	12-9
12.2.1	Example: Best Physical Standby Database for a Failover	12-10
12.2.2	Example: Best Logical Standby Database for a Failover	12-16
12.3	Configuring a Logical Standby Database to Support a New Primary Database.....	12-20
12.3.1	When the New Primary Database Was Formerly a Physical Standby Database ..	12-20
12.3.2	When the New Primary Database Was Formerly a Logical Standby Database	12-21
12.4	Using Flashback Database After a Failover	12-23
12.4.1	Flashing Back a Failed Primary Database into a Physical Standby Database	12-24
12.4.2	Flashing Back a Failed Primary Database into a Logical Standby Database	12-25
12.4.3	Flashing Back a Logical Standby Database to a Specific Applied SCN.....	12-26
12.5	Using Flashback Database After Issuing an Open Resetlogs Statement	12-27
12.5.1	Flashing Back a Physical Standby Database to a Specific Point-in-Time	12-27
12.5.2	Flash Back a Logical Standby Database After Flashing Back the Primary	12-28
12.6	Using a Physical Standby Database for Read/Write Testing and Reporting	12-29
12.7	Using RMAN Incremental Backups to Roll Forward a Physical Standby Database ...	12-33
12.7.1	Physical Standby Database Lags Far Behind the Primary Database.....	12-33
12.7.2	Physical Standby Database Has Nologging Changes On a Subset of Datafiles	12-34
12.7.3	Physical Standby Database Has Widespread Nologging Changes.....	12-36
12.8	Using a Physical Standby Database with a Time Lag	12-37
12.8.1	Establishing a Time Lag on a Physical Standby Database.....	12-37
12.8.2	Failing Over to a Physical Standby Database with a Time Lag	12-38
12.8.3	Switching Over to a Physical Standby Database with a Time Lag.....	12-38
12.9	Recovering From a Network Failure.....	12-39
12.10	Recovering After the NOLOGGING Clause Is Specified	12-40
12.10.1	Recovery Steps for Logical Standby Databases.....	12-41
12.10.2	Recovery Steps for Physical Standby Databases	12-41
12.10.3	Determining If a Backup Is Required After Unrecoverable Operations	12-43
12.11	Resolving Archive Gaps Manually	12-43
12.11.1	What Causes Archive Gaps?	12-43
12.11.1.1	Creation of the Standby Database.....	12-43
12.11.1.2	Shutdown of the Standby Database When the Primary Database Is Open	12-44
12.11.1.3	Network Failure Prevents Transmission of Redo.....	12-45
12.11.2	Determining If an Archive Gap Exists.....	12-45
12.11.3	Manually Transmitting Log Files in the Archive Gap to the Standby Site	12-46
12.11.4	Manually Applying Log Files in the Archive Gap to the Standby Database.....	12-47
12.12	Creating a Standby Database That Uses OMF or ASM.....	12-48

Part II Reference

13 Initialization Parameters

14 LOG_ARCHIVE_DEST_n Parameter Attributes

AFFIRM and NOAFFIRM.....	14-2
ALTERNATE.....	14-4
ARCH and LGWR.....	14-6
DB_UNIQUE_NAME	14-7
DELAY.....	14-8
DEPENDENCY	14-10
LOCATION and SERVICE	14-12
MANDATORY and OPTIONAL.....	14-14
MAX_CONNECTIONS.....	14-16
MAX_FAILURE.....	14-17
NET_TIMEOUT	14-19
NOREGISTER.....	14-21
REOPEN	14-22
SYNC and ASYNC.....	14-23
TEMPLATE.....	14-24
VALID_FOR.....	14-26
VERIFY	14-28

15 SQL Statements Relevant to Data Guard

15.1 ALTER DATABASE Statements	15-1
15.2 ALTER SESSION Statements	15-4

16 Views Relevant to Oracle Data Guard

Part III Appendixes

A Troubleshooting Data Guard

A.1 Common Problems	A-1
A.1.1 Standby Archive Destination Is Not Defined Properly.....	A-1
A.1.2 Renaming Datafiles with the ALTER DATABASE Statement.....	A-1
A.1.3 Standby Database Does Not Receive Redo Data from the Primary Database.....	A-2
A.1.4 You Cannot Mount the Physical Standby Database.....	A-3
A.2 Log File Destination Failures.....	A-3
A.3 Handling Logical Standby Database Failures.....	A-4
A.4 Problems Switching Over to a Standby Database.....	A-4
A.4.1 Switchover Fails Because Redo Data Was Not Transmitted	A-4
A.4.2 Switchover Fails Because SQL Sessions Are Still Active	A-5
A.4.3 Switchover Fails Because User Sessions Are Still Active.....	A-6
A.4.4 Switchover Fails with the ORA-01102 Error.....	A-6
A.4.5 Redo Data Is Not Applied After Switchover	A-7

A.4.6	Roll Back After Unsuccessful Switchover and Start Over	A-7
A.5	What to Do If SQL Apply Stops.....	A-8
A.6	Network Tuning for Redo Data Transmission	A-9
A.7	Slow Disk Performance on Standby Databases.....	A-10
A.8	Log Files Must Match to Avoid Primary Database Shutdown	A-10
A.9	Troubleshooting a Logical Standby Database	A-10
A.9.1	Recovering from Errors.....	A-10
A.9.1.1	DDL Transactions Containing File Specifications	A-11
A.9.1.2	Recovering from DML Failures.....	A-12
A.9.2	Troubleshooting SQL*Loader Sessions	A-12
A.9.3	Troubleshooting Long-Running Transactions	A-13
A.9.4	Troubleshooting ORA-1403 Errors with Flashback Transactions	A-17

B Upgrading Databases in a Data Guard Configuration

B.1	Before You Upgrade the Oracle Database Software	B-1
B.2	Upgrading Oracle Database with a Physical Standby Database In Place	B-2
B.3	Upgrading Oracle Database with a Logical Standby Database In Place	B-4

C Data Type and DDL Support on a Logical Standby Database

C.1	Data Type Considerations	C-1
C.1.1	Supported Data Types in a Logical Standby Database	C-1
C.1.2	Unsupported Data Types in a Logical Standby Database	C-2
C.2	Storage Type Considerations	C-2
C.2.1	Support Storage Types.....	C-2
C.2.2	Unsupported Storage Type	C-2
C.3	PL/SQL Supplied Packages Considerations	C-2
C.3.1	Supported PL/SQL Supplied Packages	C-3
C.3.2	Unsupported PL/SQL Supplied Packages	C-3
C.4	Unsupported Tables, Sequences, and Views	C-3
C.5	Skipped SQL Statements on a Logical Standby Database	C-4
C.6	DDL Statements Supported by a Logical Standby Database	C-5

D Data Guard and Real Application Clusters

D.1	Configuring Standby Databases in a Real Application Clusters Environment	D-1
D.1.1	Setting Up a Multi-Instance Primary with a Single-Instance Standby	D-1
D.1.2	Setting Up a Multi-Instance Primary with a Multi-Instance Standby	D-2
D.2	Configuration Considerations in a Real Application Clusters Environment	D-4
D.2.1	Format for Archived Redo Log Filenames.....	D-4
D.2.2	Archive Destination Quotas.....	D-5
D.2.3	Data Protection Modes.....	D-5
D.2.4	Role Transitions	D-5
D.2.4.1	Switchovers	D-5
D.2.4.2	Failovers.....	D-6
D.3	Troubleshooting	D-6
D.3.1	Switchover Fails in a Real Application Clusters Configuration	D-6
D.3.2	Avoiding Downtime in Real Application Clusters During a Network Outage	D-6

E Cascaded Destinations

E.1	Configuring Cascaded Destinations	E-2
E.1.1	Configuring Cascaded Destinations for Physical Standby Databases.....	E-2
E.1.2	Configuring Cascaded Destinations for Logical Standby Databases.....	E-3
E.2	Role Transitions with Cascaded Destinations	E-4
E.2.1	Standby Databases Receiving Redo Data from a Physical Standby Database	E-4
E.2.2	Standby Databases Receiving Redo Data from a Logical Standby Database	E-4
E.3	Examples of Cascaded Destinations	E-4
E.3.1	Local Physical Standby and Cascaded Remote Physical Standby.....	E-4
E.3.2	Local Physical Standby and Cascaded Remote Logical Standby	E-5
E.3.3	Local and Remote Physical Standby and Cascaded Local Logical Standby	E-5
E.3.4	Consolidated Reporting with Cascaded Logical Standby Destinations.....	E-6
E.3.5	Temporary Use of Cascaded Destinations During Network Upgrades.....	E-6

F Creating a Standby Database with Recovery Manager

F.1	Preparing to Use RMAN to Create a Standby Database.....	F-1
F.1.1	About Standby Database Preparation Using RMAN.....	F-2
F.1.2	Creating the Standby Control File with RMAN.....	F-2
F.1.3	Naming the Standby Database Datafiles When Using RMAN.....	F-3
F.1.4	Naming the Standby Database Log Files When Using RMAN	F-4
F.2	Creating a Standby Database with RMAN: Overview.....	F-5
F.2.1	RMAN Standby Creation Without Recovery	F-5
F.2.2	RMAN Standby Creation with Recovery	F-6
F.3	Setting Up the Standby Database	F-7
F.3.1	Setting Up a Standby Database When Files Are Not Oracle Managed Files	F-7
F.3.2	Setting Up a Standby Database When All Files Are Oracle Managed Files	F-8
F.3.3	Setting Up a Standby Databases When a Subset of Files Are Oracle Managed Files	F-9
F.4	Creating a Standby Database with the Same Directory Structure.....	F-9
F.4.1	Creating the Standby Database Without Performing Recovery	F-9
F.4.2	Creating the Standby Database and Performing Recovery	F-10
F.5	Creating a Standby Database with a Different Directory Structure	F-10
F.5.1	Naming Standby Database Files with DB_FILE_NAME_CONVERT	F-11
F.5.1.1	Creating the Standby Database Without Performing Recovery.....	F-11
F.5.1.2	Creating the Standby Database and Performing Recovery.....	F-11
F.5.2	Naming Standby Database Files with SET NEWNAME	F-11
F.5.2.1	Creating the Standby Database Without Performing Recovery.....	F-11
F.5.2.2	Creating the Standby Database and Performing Recovery.....	F-12
F.5.3	Naming Standby Database Files with CONFIGURE AUXNAME.....	F-13
F.5.3.1	Creating the Standby Database Without Performing Recovery.....	F-13
F.5.3.2	Creating the Standby Database and Performing Recovery.....	F-14
F.6	Creating a Standby Database on the Local Host	F-15
F.7	Creating a Standby Database with Image Copies.....	F-15
F.7.1	Overview.....	F-15
F.7.2	When Copies and Datafiles Use the Same Names.....	F-17
F.7.3	When Copies and Datafiles Use Different Names.....	F-17
F.7.3.1	Creating the Standby Database Without Performing Recovery.....	F-17
F.7.3.2	Creating the Standby Database and Performing Recovery.....	F-18

F.8	Usage Scenario	F-19
-----	----------------------	------

G Setting Archive Tracing

G.1	LOG_ARCHIVE_TRACE Initialization Parameter	G-1
G.2	Determining the Location of the Trace Files	G-1
G.2.1	Setting the LOG_ARCHIVE_TRACE Initialization Parameter	G-1
G.2.2	Choosing an Integer Value	G-2

Index

List of Examples

3-1	Adding a Standby Redo Log File Group to a Specific Thread	3-3
3-2	Adding a Standby Redo Log File Group to a Specific Group Number	3-3
3-3	Primary Database: Primary Role Initialization Parameters	3-4
3-4	Primary Database: Standby Role Initialization Parameters.....	3-5
3-5	Modifying Initialization Parameters for a Physical Standby Database	3-8
4-1	Primary Database: Logical Standby Role Initialization Parameters.....	4-4
4-2	Modifying Initialization Parameters for a Logical Standby Database	4-6
5-1	Specifying a Local Archiving Destination.....	5-3
5-2	Specifying a Remote Archiving Destination.....	5-4
5-3	Primary Database Initialization Parameters for a Shared Recovery Area.....	5-7
5-4	Standby Database Initialization Parameters for a Shared Recovery Area	5-7
5-5	Initialization Parameters for LGWR Synchronous Archival	5-11
5-6	Initialization Parameters for LGWR Asynchronous Archiving.....	5-12
5-7	Setting a Retry Time and Limit	5-18
5-8	Setting a Mandatory Archiving Destination.....	5-23
11-1	Monitoring Events with DBA_LOGSTDBY_EVENTS	11-7
12-1	Finding VALID_FOR Information in the V\$ARCHIVE_DEST View	12-9
14-1	Automatically Failing Over to an Alternate Destination	14-5
14-2	Defining an Alternate Oracle Net Service Name to the Same Standby Database	14-5
A-1	Setting a Retry Time and Limit	A-3
A-2	Specifying an Alternate Destination	A-3
A-3	Warning Messages Reported for ITL Pressure.....	A-15

List of Figures

1-1	Typical Data Guard Configuration	1-3
1-2	Automatic Updating of a Physical Standby Database	1-4
1-3	Automatic Updating of a Logical Standby Database	1-5
1-4	Data Guard Overview Page in Oracle Enterprise Manager	1-7
2-1	Possible Standby Configurations.....	2-7
5-1	Transmitting Redo Data.....	5-2
5-2	Primary Database Archiving When There Is No Standby Database	5-4
5-3	Archiving to Local Destinations Before Archiving to Remote Destinations.....	5-9
5-4	LGWR SYNC Archival to a Remote Destination with Standby Redo Log Files.....	5-12
5-5	LGWR ASYNC Archival with Network Server (LNSn) Processes.....	5-13
5-6	Archival Operation to an Alternate Destination Device	5-17
5-7	Data Guard Configuration with Dependent Destinations.....	5-25
6-1	Applying Redo Data to a Standby Destination Using Real-Time Apply	6-3
7-1	Data Guard Configuration Before Switchover	7-4
7-2	Standby Databases Before Switchover to the New Primary Database	7-4
7-3	Data Guard Environment After Switchover	7-5
7-4	Failover to a Standby Database.....	7-6
8-1	Standby Database Open for Read-Only Access.....	8-3
9-1	SQL Apply Processing	9-1
9-2	Progress States During SQL Apply Processing.....	9-10
11-1	Data Guard Configuration Before Upgrade	11-2
11-2	Upgrade the Logical Standby Database Release	11-6
11-3	Running Mixed Releases.....	11-7
11-4	After a Switchover	11-9
11-5	Both Databases Upgraded	11-10
12-1	Primary and Physical Standby Databases Before a Role Transition	12-2
12-2	Primary and Physical Standby Databases After a Role Transition.....	12-3
12-3	Configuring Destinations for a Primary Database and a Logical Standby Database ...	12-4
12-4	Primary and Logical Standby Databases After a Role Transition	12-6
12-5	Configuring a Primary Database with Physical and Logical Standby Databases.....	12-7
12-6	Primary, Physical, and Logical Standby Databases After a Role Transition.....	12-8
12-7	Using a Physical Standby Database As a Testing and Reporting Database.....	12-30
12-8	Manual Recovery of Archived Redo Log Files in an Archive Gap.....	12-44
D-1	Transmitting Redo Data from a Multi-Instance Primary Database.....	D-2
D-2	Standby Database in Real Application Clusters.....	D-3
E-1	Cascaded Destination Configuration Example	E-1

List of Tables

2-1	Standby Database Location and Directory Options	2-7
3-1	Preparing the Primary Database for Physical Standby Database Creation.....	3-1
3-2	Creating a Physical Standby Database.....	3-7
4-1	Preparing the Primary Database for Logical Standby Database Creation	4-1
4-2	Creating a Logical Standby Database	4-3
5-1	LOG_ARCHIVE_DEST_STATE_n Initialization Parameter Attributes	5-3
5-2	Minimum Requirements for Data Protection Modes	5-19
5-3	Wait Events for Destinations Configured with the ARCH Attribute	5-30
5-4	Wait Events for Destinations Configured with the LGWR SYNC Attributes.....	5-31
5-5	Wait Events for Destinations Configured with the LGWR ASYNC Attributes	5-31
8-1	Actions Required on a Standby Database After Changes to a Primary Database	8-6
8-2	Location Where Common Actions on the Primary Database Can Be Monitored	8-14
11-1	Step-by-Step Procedure to Upgrade Oracle Database Software	11-5
12-1	Data Guard Scenarios.....	12-1
12-2	Initialization Parameter Settings for Primary and Physical Standby Databases.....	12-2
12-3	Initialization Parameter Settings for Primary and Logical Standby Databases.....	12-4
12-4	Initialization Parameters for Primary, Physical, and Logical Standby Databases	12-7
12-5	Identifiers for the Physical Standby Database Example.....	12-10
12-6	Identifiers for Logical Standby Database Example.....	12-16
13-1	Initialization Parameters for Instances in a Data Guard Configuration	13-1
14-1	Directives for the TEMPLATE Attribute	14-24
14-2	VALID_FOR Attribute Values	14-27
15-1	ALTER DATABASE Statements Used in Data Guard Environments	15-1
15-2	ALTER SESSION Statement Used in Data Guard Environments	15-4
16-1	Views That Are Pertinent to Data Guard Configurations	16-1
A-1	Common Processes That Prevent Switchover	A-6
A-2	Fixing Typical SQL Apply Errors	A-9
C-1	Values for stmt Parameter of the DBMS_LOGSTDBY.SKIP procedure	C-5
C-2	Statement Options for Skipping SQL DDL Statements.....	C-7
D-1	Directives for the LOG_ARCHIVE_FORMAT Initialization Parameter	D-4
E-1	Initialization Parameters for Primary, Physical, and Logical Standby Databases	E-3
F-1	Standby Database Preparation Using RMAN	F-2
F-2	Order of Precedence for Naming Datafiles in Standby Database.....	F-4
F-3	Using Image Copies to Create a Standby Database: Scenario.....	F-16

Preface

Oracle Data Guard is the most effective solution available today to protect the core asset of any enterprise—its data, and make it available on a 24x7 basis even in the face of disasters and other calamities. This guide describes Oracle Data Guard technology and concepts, and helps you configure and implement standby databases.

Audience

Oracle Data Guard Concepts and Administration is intended for database administrators (DBAs) who administer the backup, restoration, and recovery operations of an Oracle database system.

To use this document, you should be familiar with relational database concepts and basic backup and recovery administration. You should also be familiar with the operating system environment under which you are running Oracle software.

Documentation Accessibility

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

<http://www.oracle.com/accessibility/>

Accessibility of Code Examples in Documentation

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

Accessibility of Links to External Web Sites in Documentation

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

TTY Access to Oracle Support Services

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

Related Documents

Readers of *Oracle Data Guard Concepts and Administration* should also read:

- The beginning of *Oracle Database Concepts*, that provides an overview of the concepts and terminology related to the Oracle database and serves as a foundation for the more detailed information in this guide.
- The chapters in the *Oracle Database Administrator's Guide* that deal with managing the control files, online redo log files, and archived redo log files.
- The chapter in the *Oracle Database Utilities* that discusses LogMiner technology.
- *Oracle Data Guard Broker* that describes the graphical user interface and command-line interface for automating and centralizing the creation, maintenance, and monitoring of Data Guard configurations.
- Oracle Enterprise Manager online Help system

Discussions in this book also refer you to the following guides:

- *Oracle Database SQL Reference*
- *Oracle Database Reference*
- *Oracle Database Backup and Recovery Basics*
- *Oracle Database Backup and Recovery Advanced User's Guide*
- *Oracle Database Net Services Administrator's Guide*
- *SQL*Plus User's Guide and Reference*
- *Oracle Database High Availability Overview*

Also, see *Oracle Streams Concepts and Administration* for information about Oracle Streams and the Streams Downstream Capture Database. The Streams downstream capture process uses the Oracle Data Guard redo transport services to transfer redo data to log files on a remote database where a Streams capture process captures changes in the archived redo log files at the remote destination.

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.
<i>italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
monospace	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

What's New in Oracle Data Guard?

This preface describes the new features added to Oracle Data Guard in release 10.2 and provides links to additional information. The features and enhancements described in this preface were added to Oracle Data Guard in 10g Release 2 (10.2). The new features are described under the following main areas:

- [New Features Common to Redo Apply and SQL Apply](#)
- [New Features Specific to Redo Apply and Physical Standby Databases](#)
- [New Features Specific to SQL Apply and Logical Standby Databases](#)

New Features Common to Redo Apply and SQL Apply

The following enhancements to Oracle Data Guard in 10g Release 2 (10.2) improve ease-of-use, manageability, performance, and include innovations that improve disaster-recovery capabilities:

- **Fast-start failover**

Fast-start failover provides the ability to automatically, quickly, and reliably fail over to a designated, synchronized standby database in the event of loss of the primary database, without requiring that you perform complex manual steps to invoke the failover.

Also, after a fast-start failover occurs, the old primary database is automatically reconfigured as a new standby database upon reconnection to the configuration. This enables Data Guard to restore disaster protection in the configuration easily, without complex manual steps, improving the robustness of the disaster-recovery features of Data Guard, as well as improving Data Guard manageability.

These new capabilities allow you to maintain uptime and increase the availability, as well the robustness of disaster recovery. Plus, there is less need for manual intervention, thereby reducing management costs.

See Also: *Oracle Data Guard Broker*

- **Flashback Database across Data Guard switchovers**

It is now possible to flash back the primary and standby databases to an SCN or a point in time prior to a switchover operation. When you use this feature of Flashback Database on a physical standby database, the standby role is preserved. On a logical standby database, the role of the standby database is changed to what it was at the target SCN or time.

This feature extends the flashback window, providing more flexibility to detect and correct human errors.

See Also: [Section 7.4, "Using Flashback Database After a Role Transition"](#)

- **Asynchronous Redo Transmission**

Asynchronous redo transmission using the log writer process (LGWR ASYNC) has been improved to reduce the performance impact on the primary database. During asynchronous redo transmission, the network server (LNS n) process transmits redo data out of the online redo log files on the primary database and no longer interacts directly with the log writer process.

This change in behavior allows the log writer process to write redo data to the current online redo log file and continue processing the next request without waiting for interprocess communication or network I/O to complete.

See Also: [Section 5.3.2.3, "LGWR ASYNC Archival Processing"](#) and the [SYNC](#) and [ASYNC](#) attributes in [Chapter 14](#)

- **New `MAX_CONNECTIONS` attribute on the `LOG_ARCHIVE_DEST_n` parameter**

This attribute specifies how the archiver (ARC n) processes on the primary database coordinate when sending redo data to standby databases. If the `MAX_CONNECTIONS` attribute is set to a nonzero value, redo transport services use multiple network connections to transmit redo data using archiver processes.

See Also: ["MAX_CONNECTIONS" attribute](#) on page 14-16

- **Data Guard enhancements in Oracle Enterprise Manager**

If you use the Data Guard broker to manage your Data Guard configuration, you can take advantage of the following enhancements in Oracle Enterprise Manager.

- Estimated Failover Time (seconds)

The approximate number of seconds required to fail over to this standby database. This accounts for the startup time (if necessary) plus the remaining time it would require to apply all the available redo data on the standby database. If it is not necessary to start the database, this metric shows only the remaining *apply* time.

- Apply Lag (seconds)

The number of seconds that the standby database is behind the primary database in applying redo data.

- Redo Generation Rate (KB/second)

Displays the amount of redo generation rate in KB/second on the primary database.

- Redo Apply Rate (KB/second)

Displays the Redo Apply Rate in KB/second on this standby database.

- Transport Lag (seconds)

The approximate number of seconds of redo data not yet available on this standby database. This may be because the redo has not yet been shipped or there may be a gap.

- Data Guard Status

Use the Data Guard Status metric to check the status of each database in the Data Guard configuration.

By default, a critical and warning threshold value was set for this metric column. Alerts will be generated when threshold values are reached. You can edit the value for a threshold, as required.

– Fast-Start Failover Occurred

When fast-start failover is enabled, this metric generates a critical alert on the new primary database (old standby database) if a fast-start failover occurs. The fast-start failover SCN must be initialized to a value before the metric will alert. This usually takes one collection interval. Once a fast-start failover occurs and the new primary database is ready, the fast-start failover alert is generated. The alert is cleared after one collection interval. A critical alert is configured by default.

- * Both primary and standby databases must be configured with `SYSDBA` monitoring access.
- * Shows the time when a fast-start failover occurred: the value is zero if fast-start failover has not occurred and one if fast-start failover occurred.

– Fast-Start Failover SCN

When fast-start failover is enabled, this metric generates a critical alert on the new primary database (old standby database) if a fast-start failover occurs. The fast-start failover SCN must be initialized to a value before the metric will alert. This usually takes one collection interval. Once a fast-start failover occurs and the new primary database is ready, the fast-start failover alert is generated. The alert is cleared after one collection interval. A critical alert is configured by default.

- * Both primary and standby databases must be configured with `SYSDBA` monitoring access.
- * Any value indicates the metric is ready to trigger.

– Fast-Start Failover Time

When fast-start failover is enabled, this metric will generate a critical alert on the new primary database (old standby database) if a fast-start failover occurs. The fast-start failover SCN must be initialized to a value before the metric will create an alert. This usually takes one collection interval. Once a fast-start failover occurs and the new primary database is ready, the fast-start failover alert fires. It then clears after one collection interval. A critical alert is configured by default.

- * Both primary and standby databases must be configured with `SYSDBA` monitoring access.
- * A time stamp displays if fast-start failover occurred.

See Also: Oracle Enterprise Manager online Help system

- New support for Change Data Capture and Streams:
 - Distributed (heterogeneous) Asynchronous Change Data Capture
 - Downstream Capture Hot Mining

See Also: *Oracle Streams Concepts and Administration* and *Oracle Database Data Warehousing Guide* (for Change Data Capture information)

New Features Specific to Redo Apply and Physical Standby Databases

The following list summarizes the new features that are specific to Redo Apply and physical standby databases in Oracle Database 10g Release 2 (10.2):

- **Faster Redo Apply failover**

Allows you to transition a physical standby database to the primary database role without doing a database restart, as long as the standby database has not been opened read-only since the last time it was started.

This enables faster recovery from a failure or outage, increasing the availability of the system.

See Also: [Section 7.2.2, "Failovers Involving a Physical Standby Database"](#)

- **Easy conversion of a physical standby database to a reporting database**

A physical standby database can be activated as a primary database, opened read/write for reporting purposes, and then flashed back to a point in the past to be easily converted back to a physical standby database. At this point, Data Guard automatically synchronizes the standby database with the primary database. This allows the physical standby database to be utilized for reporting and read/write cloning activities.

See Also: [Section 12.6, "Using a Physical Standby Database for Read/Write Testing and Reporting"](#)

- **New FORCE keyword on RECOVER MANAGED STANDBY DATABASE FINISH**

The new FORCE option on the ALTER DATABASE RECOVER MANAGED STANDBY DATABASE FINISH statement stops active RFS processes on the target standby database so the failover will proceed immediately, as soon as the logs have been applied.

See Also: [Section 7.2.2, "Failovers Involving a Physical Standby Database"](#) and the ALTER DATABASE syntax in *Oracle Database SQL Reference*

- **RMAN automatically re-creates tempfiles after recovery**

Temporary datafiles that belong to locally managed temporary tablespaces are automatically created during the recovery operation by RMAN, thus it is no longer necessary to create and associate a tempfile with a temporary tablespace on a physical standby database.

- **Miscellaneous changes improve usage and manageability**

By deprecating unnecessary initialization parameters and certain SQL statement clauses and keywords, the usage and manageability of physical standby databases has been improved.

See Also: The SQL statements relevant to Data Guard in *Oracle Database SQL Reference* and the LOG_ARCHIVE_DEST_n initialization parameter in *Oracle Database Reference*

New Features Specific to SQL Apply and Logical Standby Databases

The following list summarizes the new features for SQL Apply and logical standby databases in Oracle Database 10g Release 2 (10.2):

- **Faster SQL Apply failover**

Failover to a logical standby database can now be completed much faster because it is no longer necessary to restart SQL Apply as a part of the failover operation. This enables the Data Guard configuration to recover from a failure or outage much faster, increasing the availability of the system.

See Also: [Section 7.3.2, "Failovers Involving a Logical Standby Database"](#)

- **Additional datatype support for Index Organized Tables**

SQL Apply now supports mining of redo records generated by index organized tables containing LOB columns and overflow segments.

See Also: [Section 4.1.1, "Determine Support for Data Types and Storage Attributes for Tables"](#)

- **Automatic deletion of applied archived redo log files**

Archived log files, once they are applied on the logical standby database, will be automatically deleted by SQL Apply. This reduces storage consumption on the logical standby database and improves Data Guard manageability.

See Also: [Section 9.3.2, "Automatic Deletion of Log Files"](#)

- **Optimized creation of logical standby database**

Creation of a logical standby database no longer requires the creation of a specialized logical standby control file, which could not be used by RMAN. Logical standby databases can now be created easily from a physical standby database. This reduces the specialized manual operations for creating a logical standby database and improves Data Guard manageability.

See Also: [Section 4.2, "Step-by-Step Instructions for Creating a Logical Standby Database"](#)

- **Added several enhancements for managing logical standby databases:**

- New views:

- * V\$LOGSTDBY_PROCESS (replaces the deprecated V\$LOGSTDBY view)
- * V\$LOGSTDBY_STATE
- * V\$LOGSTDBY_PROGRESS
- * V\$LOGSTDBY_TRANSACTION
- * V\$DATAGUARD_STATS

- New DBMS_LOGSTDBY.REBUILD () subprogram on the DBMS_LOGSTDBY PL/SQL package

- Tracing

See Also: [Chapter 9, "Managing a Logical Standby Database"](#)

Part I

Concepts and Administration

This part contains the following chapters:

- Chapter 1, "Introduction to Oracle Data Guard"
- Chapter 2, "Getting Started with Data Guard"
- Chapter 3, "Creating a Physical Standby Database"
- Chapter 4, "Creating a Logical Standby Database"
- Chapter 5, "Redo Transport Services"
- Chapter 6, "Log Apply Services"
- Chapter 7, "Role Transitions"
- Chapter 8, "Managing a Physical Standby Database"
- Chapter 9, "Managing a Logical Standby Database"
- Chapter 10, "Using RMAN to Back Up and Restore Files"
- Chapter 11, "Using SQL Apply to Upgrade the Oracle Database"
- Chapter 12, "Data Guard Scenarios"

Introduction to Oracle Data Guard

Oracle Data Guard ensures high availability, data protection, and disaster recovery for enterprise data. Data Guard provides a comprehensive set of services that create, maintain, manage, and monitor one or more standby databases to enable production Oracle databases to survive disasters and data corruptions. Data Guard maintains these standby databases as transactionally consistent copies of the production database. Then, if the production database becomes unavailable because of a planned or an unplanned outage, Data Guard can switch any standby database to the production role, minimizing the downtime associated with the outage. Data Guard can be used with traditional backup, restoration, and cluster techniques to provide a high level of data protection and data availability.

With Data Guard, administrators can optionally improve production database performance by offloading resource-intensive backup and reporting operations to standby systems.

This chapter includes the following topics that describe the highlights of Oracle Data Guard:

- [Data Guard Configurations](#)
- [Data Guard Services](#)
- [Data Guard Broker](#)
- [Data Guard Protection Modes](#)
- [Data Guard and Complementary Technologies](#)
- [Summary of Data Guard Benefits](#)

1.1 Data Guard Configurations

A **Data Guard configuration** consists of one production database and one or more standby databases. The databases in a Data Guard configuration are connected by Oracle Net and may be dispersed geographically. There are no restrictions on where the databases are located, provided they can communicate with each other. For example, you can have a standby database on the same system as the production database, along with two standby databases on other systems at remote locations.

You can manage primary and standby databases using the SQL command-line interfaces or the Data Guard broker interfaces, including a command-line interface (DGMGRL) and a graphical user interface that is integrated in Oracle Enterprise Manager.

1.1.1 Primary Database

A Data Guard configuration contains one production database, also referred to as the primary database, that functions in the primary role. This is the database that is accessed by most of your applications.

The primary database can be either a single-instance Oracle database or an Oracle Real Application Clusters database.

1.1.2 Standby Databases

A standby database is a transactionally consistent copy of the primary database. Using a backup copy of the primary database, you can create up to nine standby databases and incorporate them in a Data Guard configuration. Once created, Data Guard automatically maintains each standby database by transmitting redo data from the primary database and then applying the redo to the standby database.

Similar to a primary database, a standby database can be either a single-instance Oracle database or an Oracle Real Application Clusters database.

A standby database can be either a physical standby database or a logical standby database:

- **Physical standby database**

Provides a physically identical copy of the primary database, with on disk database structures that are identical to the primary database on a block-for-block basis. The database schema, including indexes, are the same. A physical standby database is kept synchronized with the primary database, though **Redo Apply**, which recovers the redo data received from the primary database and applies the redo to the physical standby database.

A physical standby database can be used for business purposes other than disaster recovery on a limited basis.

- **Logical standby database**

Contains the same logical information as the production database, although the physical organization and structure of the data can be different. The logical standby database is kept synchronized with the primary database though **SQL Apply**, which transforms the data in the redo received from the primary database into SQL statements and then executing the SQL statements on the standby database.

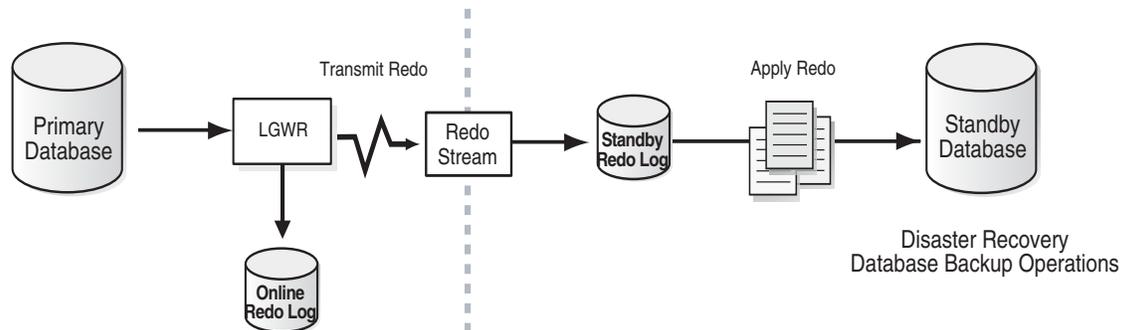
A logical standby database can be used for other business purposes in addition to disaster recovery requirements. This allows users to access a logical standby database for queries and reporting purposes at any time. Also, using a logical standby database, you can upgrade Oracle Database software and patch sets with almost no downtime. Thus, a logical standby database can be used concurrently for data protection, reporting, and database upgrades.

1.1.3 Configuration Example

Figure 1–1 shows a typical Data Guard configuration that contains a primary database that transmits redo data to a standby database. The standby database is remotely located from the primary database for disaster recovery and backup operations. You can configure the standby database at the same location as the primary database. However, for disaster recovery purposes, Oracle recommends you configure standby databases at remote locations.

Figure 1–1 shows a typical Data Guard configuration in which redo is being applied out of standby redo log files to a standby database.

Figure 1–1 Typical Data Guard Configuration



1.2 Data Guard Services

The following sections explain how Data Guard manages the transmission of redo data, the application of redo data, and changes to the database roles:

- [Redo Transport Services](#)
Control the automated transfer of redo data from the production database to one or more archival destinations.
- [Log Apply Services](#)
Apply redo data on the standby database to maintain transactional synchronization with the primary database. Redo data can be applied either from archived redo log files, or, if real-time apply is enabled, directly from the standby redo log files as they are being filled, without requiring the redo data to be archived first at the standby database.
- [Role Transitions](#)
Change the role of a database from a standby database to a primary database, or from a primary database to a standby database using either a switchover or a failover operation.

1.2.1 Redo Transport Services

Redo transport services control the automated transfer of redo data from the production database to one or more archival destinations.

Redo transport services perform the following tasks:

- Transmit redo data from the primary system to the standby systems in the configuration
- Manage the process of resolving any gaps in the archived redo log files due to a network failure
- Enforce the database protection modes (described in [Section 1.4](#))
- Automatically detect missing or corrupted archived redo log files on a standby system and automatically retrieve replacement archived redo log files from the primary database or another standby database

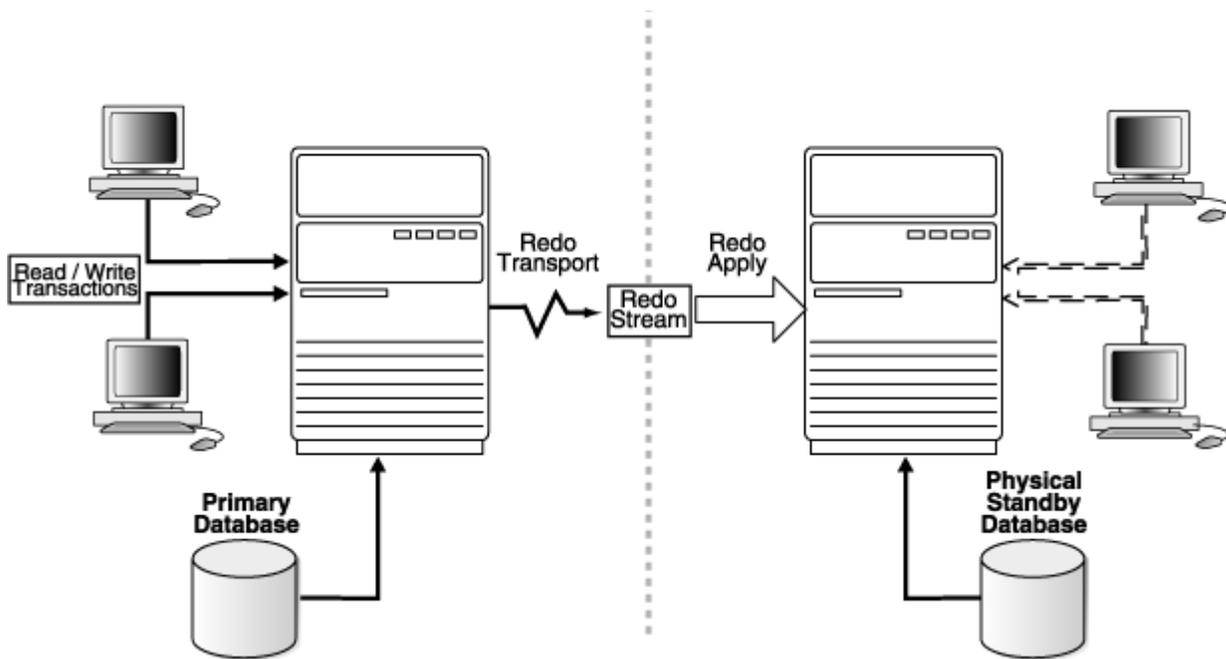
1.2.2 Log Apply Services

The redo data transmitted from the primary database is written on the standby system into standby redo log files, if configured, and then archived into archived redo log files. **Log apply services** automatically apply the redo data on the standby database to maintain consistency with the primary database. It also allows read-only access to the data.

The main difference between physical and logical standby databases is the manner in which log apply services apply the archived redo data:

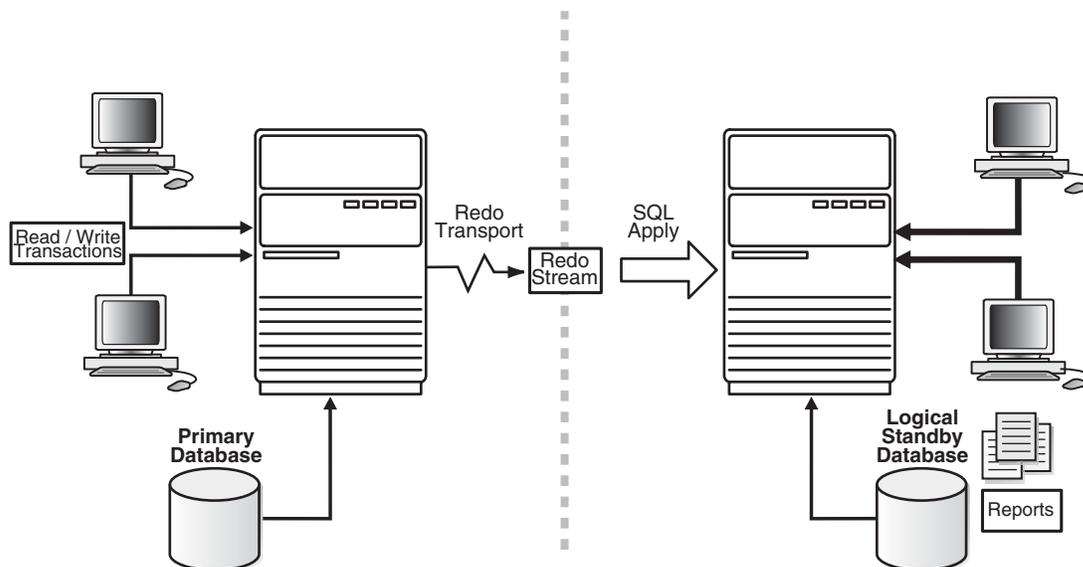
- For physical standby databases, Data Guard uses **Redo Apply** technology, which applies redo data on the standby database using standard recovery techniques of an Oracle database, as shown in [Figure 1-2](#).

Figure 1-2 Automatic Updating of a Physical Standby Database



- For logical standby databases, Data Guard uses **SQL Apply** technology, which first transforms the received redo data into SQL statements and then executes the generated SQL statements on the logical standby database, as shown in [Figure 1-3](#).

Figure 1-3 Automatic Updating of a Logical Standby Database



1.2.3 Role Transitions

An Oracle database operates in one of two roles: primary or standby. Using Data Guard, you can change the role of a database using either a switchover or a failover operation.

A **switchover** is a role reversal between the primary database and one of its standby databases. A switchover ensures no data loss. This is typically done for planned maintenance of the primary system. During a switchover, the primary database transitions to a standby role, and the standby database transitions to the primary role. The transition occurs without having to re-create either database.

A **failover** is when the primary database is unavailable. Failover is performed only in the event of a catastrophic failure of the primary database, and the failover results in a transition of a standby database to the primary role. The database administrator can configure Data Guard to ensure no data loss.

The role transitions described in this documentation are invoked manually using SQL statements. You can also use the Oracle Data Guard broker to simplify role transitions and automate failovers using Oracle Enterprise Manager or the DGMGRL command-line interface, as described in [Section 1.3](#).

1.3 Data Guard Broker

The Data Guard broker is a distributed management framework that automates the creation, maintenance, and monitoring of Data Guard configurations. You can use either the Oracle Enterprise Manager graphical user interface (GUI) or the Data Guard command-line interface (DGMGRL) to:

- Create and enable Data Guard configurations, including setting up redo transport services and log apply services
- Manage an entire Data Guard configuration from any system in the configuration
- Manage and monitor Data Guard configurations that contain Real Application Clusters primary or standby databases

- Simplify switchovers and failovers by allowing you to invoke them using either a single key click in Oracle Enterprise Manager or a single command in the DGMGRL command-line interface.
- Enable fast-start failover to fail over *automatically* when the primary database becomes unavailable. When fast-start failover is enabled, the Data Guard broker determines if a failover is necessary and initiates the failover to the specified target standby database automatically, with no need for DBA intervention and with no loss of data.

In addition, Oracle Enterprise Manager automates and simplifies:

- Creating a physical or logical standby database from a backup copy of the primary database
- Adding new or existing standby databases to an existing Data Guard configuration
- Monitoring log apply rates, capturing diagnostic information, and detecting problems quickly with centralized monitoring, testing, and performance tools

See Also: *Oracle Data Guard Broker* for more information

1.3.1 Using Oracle Enterprise Manager

Oracle Enterprise Manager, also referred to as Enterprise Manager, provides a web-based interface for viewing, monitoring, and administering primary and standby databases in a Data Guard configuration. Enterprise Manager's easy-to-use interfaces combined with the broker's centralized management and monitoring of the Data Guard configuration enhance the Data Guard solution for high availability, site protection, and data protection of an enterprise.

From the Enterprise Manager Central Console, all management operations can be performed locally or remotely. You can view home pages for Oracle databases, including primary and standby databases and instances, create or add existing standby databases, start and stop instances, monitor instance performance, view events, schedule jobs, and perform backup and recovery operations. See *Oracle Data Guard Broker* and the Oracle Enterprise Manager online help system.

[Figure 1–4](#) shows the Data Guard management overview page in Enterprise Manager.

Figure 1–4 Data Guard Overview Page in Oracle Enterprise Manager

ORACLE Enterprise Manager 10g
Grid Control

Setup Preferences Help Logout

Home Targets Deployments Alerts Policies Jobs Reports

Databases Hosts Web Applications Services Systems Groups All Targets

Host: north.foo.com > Database Instance: North_Sales > Logged in As SYS

Data Guard

Page Refreshed May 26, 2005 3:05:15 PM EDT View Data Real Time: Manual Refresh

Overview

Data Guard Status **✓ Normal**
 Protection Mode **Maximum Performance**
 Fast-Start Failover **Disabled**

Primary Database

Name **North_Sales**
 Host **north**
 Data Guard Status **✓ Normal**
 Current Log **32**
 Properties **Edit**

Standby Progress Summary

The transport lag is the time difference between the primary last update and the standby last received redo. The apply lag is the time difference between the primary last update and the standby last applied redo.

Category	Value (seconds)
Transport Lag	4
Apply Lag	12

Standby Databases

Edit Remove Switchover Failover Add Standby Database

Select	Name	Host	Data Guard Status	Role	Last Received Log	Last Applied Log	Estimated Failover Time
<input checked="" type="radio"/>	DR_Sales	north	✓ Normal	Physical Standby	31	31	0 seconds

Performance

Performance Overview
Log File Details

Additional Administration

Verify
Remove Data Guard Configuration

1.3.2 Using the Data Guard Command-Line Interface

The Data Guard command-line interface (DGMGRL) enables you to control and monitor a Data Guard configuration from the DGMGRL prompt or within scripts. You can perform most of the activities required to manage and monitor the databases in the configuration using DGMGRL. See *Oracle Data Guard Broker* for complete DGMGRL reference information and examples.

1.4 Data Guard Protection Modes

In some situations, a business cannot afford to lose data. In other situations, the availability of the database may be more important than the loss of data. Some applications require maximum database performance and can tolerate some small amount of data loss. The following descriptions summarize the three distinct modes of data protection.

Maximum protection This protection mode ensures that no data loss will occur if the primary database fails. To provide this level of protection, the redo data needed to recover each transaction must be written to both the local online redo log and to the standby redo log on at least one standby database before the transaction commits. To ensure data loss cannot occur, the primary database shuts down if a fault prevents it from writing its redo stream to the standby redo log of at least one transactionally consistent standby database.

Maximum availability This protection mode provides the highest level of data protection that is possible without compromising the availability of the primary

database. Like maximum protection mode, a transaction will not commit until the redo needed to recover that transaction is written to the local online redo log and to the standby redo log of at least one transactionally consistent standby database. Unlike maximum protection mode, the primary database does not shut down if a fault prevents it from writing its redo stream to a remote standby redo log. Instead, the primary database operates in maximum performance mode until the fault is corrected, and all gaps in redo log files are resolved. When all gaps are resolved, the primary database automatically resumes operating in maximum availability mode.

This mode ensures that no data loss will occur if the primary database fails, but only if a second fault does not prevent a complete set of redo data from being sent from the primary database to at least one standby database.

Maximum performance This protection mode (the default) provides the highest level of data protection that is possible without affecting the performance of the primary database. This is accomplished by allowing a transaction to commit as soon as the redo data needed to recover that transaction is written to the local online redo log. The primary database's redo data stream is also written to at least one standby database, but that redo stream is written asynchronously with respect to the transactions that create the redo data.

When network links with sufficient bandwidth are used, this mode provides a level of data protection that approaches that of maximum availability mode with minimal impact on primary database performance.

The maximum protection and maximum availability modes require that standby redo log files are configured on at least one standby database in the configuration. All three protection modes require that specific log transport attributes be specified on the `LOG_ARCHIVE_DEST_n` initialization parameter to send redo data to at least one standby database. See [Section 5.6](#) for complete information about the data protection modes.

1.5 Data Guard and Complementary Technologies

Oracle Database provides several unique technologies that complement Data Guard to help keep business critical systems running with greater levels of availability and data protection than when using any one solution by itself. The following list summarizes some Oracle high-availability technologies:

- Oracle Real Application Clusters (RAC)

RAC enables multiple independent servers that are linked by an interconnect to share access to an Oracle database, providing high availability, scalability, and redundancy during failures. RAC and Data Guard together provide the benefits of both system-level, site-level, and data-level protection, resulting in high levels of availability and disaster recovery without loss of data:

- RAC addresses system failures by providing rapid and automatic recovery from failures, such as node failures and instance crashes. It also provides increased scalability for applications.
- Data Guard addresses site failures and data protection through transactionally consistent primary and standby databases that do not share disks, enabling recovery from site disasters and data corruption.

Many different architectures using RAC and Data Guard are possible depending on the use of local and remote sites and the use of nodes and a combination of logical and physical standby databases. See [Appendix D, "Data Guard and Real Application Clusters"](#) and *Oracle Database High Availability Overview* for RAC and Data Guard integration.

- Flashback Database

The Flashback Database feature provides fast recovery from logical data corruption and user errors. By allowing you to flash back in time, previous versions of business information that might have been erroneously changed or deleted can be accessed once again. This feature:

- Eliminates the need to restore a backup and roll forward changes up to the time of the error or corruption. Instead, Flashback Database can *roll back* an Oracle database to a previous point-in-time, without restoring datafiles.
- Provides an alternative to delaying the application of redo to protect against user errors or logical corruptions. Therefore, standby databases can be more closely synchronized with the primary database, thus reducing failover and switchover times.
- Avoids the need to completely re-create the original primary database after a failover. The failed primary database can be flashed back to a point in time before the failover and converted to be a standby database for the new primary database.

See *Oracle Database Backup and Recovery Advanced User's Guide* for information about Flashback Database, and [Section 6.2.2](#) for information delaying the application of redo data.

- Recovery Manager (RMAN)

RMAN is an Oracle utility that simplifies backing up, restoring, and recovering database files. Like Data Guard, RMAN is a feature of the Oracle database and does not require separate installation. Data Guard is well integrated with RMAN, allowing you to:

- Use the Recovery Manager `DUPLICATE` command to create a standby database from backups of your primary database.
- Take backups on a physical standby database instead of the production database, relieving the load on the production database and enabling efficient use of system resources on the standby site. Moreover, backups can be taken while the physical standby database is applying redo.
- Help manage archived redo log files by automatically deleting the archived redo log files used for input after performing a backup.

See [Appendix F, "Creating a Standby Database with Recovery Manager"](#) and *Oracle Database Backup and Recovery Basics*.

1.6 Summary of Data Guard Benefits

Data Guard offers these benefits:

- Disaster recovery, data protection, and high availability

Data Guard provides an efficient and comprehensive disaster recovery and high availability solution. Easy-to-manage switchover and failover capabilities allow role reversals between primary and standby databases, minimizing the downtime of the primary database for planned and unplanned outages.

- Complete data protection

Data Guard can ensure no data loss, even in the face of unforeseen disasters. A standby database provides a safeguard against data corruption and user errors. Storage level physical corruptions on the primary database do not propagate to

the standby database. Similarly, logical corruptions or user errors that cause the primary database to be permanently damaged can be resolved. Finally, the redo data is validated when it is applied to the standby database.

- Efficient use of system resources

The standby database tables that are updated with redo data received from the primary database can be used for other tasks such as backups, reporting, summations, and queries, thereby reducing the primary database workload necessary to perform these tasks, saving valuable CPU and I/O cycles. With a logical standby database, users can perform normal data manipulation on tables in schemas that are not updated from the primary database. A logical standby database can remain open while the tables are updated from the primary database, and the tables are simultaneously available for read-only access. Finally, additional indexes and materialized views can be created on the maintained tables for better query performance and to suit specific business requirements.

- Flexibility in data protection to balance availability against performance requirements

Oracle Data Guard offers maximum protection, maximum availability, and maximum performance modes to help enterprises balance data availability against system performance requirements.

- Automatic gap detection and resolution

If connectivity is lost between the primary and one or more standby databases (for example, due to network problems), redo data being generated on the primary database cannot be sent to those standby databases. Once a connection is reestablished, the missing archived redo log files (referred to as a gap) are automatically detected by Data Guard, which then automatically transmits the missing archived redo log files to the standby databases. The standby databases are synchronized with the primary database, without manual intervention by the DBA.

- Centralized and simple management

The Data Guard broker provides a graphical user interface and a command-line interface to automate management and operational tasks across multiple databases in a Data Guard configuration. The broker also monitors all of the systems within a single Data Guard configuration.

- Integration with Oracle Database

Data Guard is a feature of Oracle Database Enterprise Edition and does not require separate installation.

- Automatic role transitions

When fast-start failover is enabled, the Data Guard broker automatically fails over to a synchronized standby site in the event of a disaster at the primary site, requiring no intervention by the DBA. In addition, applications are automatically notified of the role transition.

Getting Started with Data Guard

A Data Guard configuration contains a primary database and up to nine associated standby databases. This chapter describes the following considerations for getting started with Data Guard:

- [Standby Database Types](#)
- [User Interfaces for Administering Data Guard Configurations](#)
- [Data Guard Operational Prerequisites](#)
- [Standby Database Directory Structure Considerations](#)
- [Online Redo Logs, Archived Redo Logs, and Standby Redo Logs](#)

2.1 Standby Database Types

A **standby database** is a transactionally consistent copy of an Oracle production database that is initially created from a backup copy of the primary database. Once the standby database is created and configured, Data Guard automatically maintains the standby database by transmitting primary database redo data to the standby system, where the redo data is applied to the standby database.

A standby database can be one of two types: a physical standby database or a logical standby database. If needed, either type of standby database can assume the role of the primary database and take over production processing. A Data Guard configuration can include physical standby databases, logical standby databases, or a combination of both types.

2.1.1 Physical Standby Databases

A physical standby database is physically identical to the primary database, with on disk database structures that are identical to the primary database on a block-for-block basis. The database schema, including indexes, are identical.

Data Guard maintains a physical standby database by performing Redo Apply. When it is not performing recovery, a physical standby database can be open in read-only mode, or it can be opened temporarily in read/write mode if Flashback Database is enabled.

- **Redo Apply**

The physical standby database is maintained by applying redo data from the archived redo log files or directly from standby redo log files on the standby system using the Oracle recovery mechanism. The recovery operation applies changes in redo blocks to data block using the data-block address. The database cannot be opened while redo is being applied.

- **Open read-only**

A physical standby database can be open in read-only mode so that you can execute queries on the database. While opened in read-only mode, the standby database can continue to receive redo data, but application of the redo data from the log files is deferred until the database resumes Redo Apply.

Although the physical standby database cannot perform both Redo Apply and be opened in read-only mode at the same time, you can switch between them. For example, you can perform Redo Apply, then open it in read-only mode for applications to run reports, and then change it back to perform Redo Apply to apply any outstanding archived redo log files. You can repeat this cycle, alternating between Redo Apply and read-only, as necessary.

The physical standby database is available to perform backups. Furthermore, the physical standby database will continue to receive redo data even if archived redo log files or standby redo log files are not being applied at that moment.

- **Open read/write**

A physical standby database can also be opened for read/write access for purposes such as creating a clone database or for read/write reporting. While opened in read/write mode, the standby database does not receive redo data from the primary database and cannot provide disaster protection.

The physical standby database can be opened temporarily in read/write mode for development, reporting, or testing purposes, and then flashed back to a point in the past to be reverted back to a physical standby database. When the database is flashed back, Data Guard automatically synchronizes the standby database with the primary database, without the need to re-create the physical standby database from a backup copy of the primary database.

See Also: [Section 12.6](#) for usage examples

Benefits of a Physical Standby Database

A physical standby database provides the following benefits:

- **Disaster recovery and high availability**

A physical standby database enables a robust and efficient disaster recovery and high availability solution. Easy-to-manage switchover and failover capabilities allow easy role reversals between primary and physical standby databases, minimizing the downtime of the primary database for planned and unplanned outages.

- **Data protection**

Using a physical standby database, Data Guard can ensure no data loss, even in the face of unforeseen disasters. A physical standby database supports all datatypes, and all DDL and DML operations that the primary database can support. It also provides a safeguard against data corruptions and user errors. Storage level physical corruptions on the primary database do not propagate to the standby database. Similarly, logical corruptions or user errors that cause the primary database to be permanently damaged can be resolved. Finally, the redo data is validated when it is applied to the standby database.

- **Reduction in primary database workload**

Oracle Recovery Manager (RMAN) can use physical standby databases to off-load backups from the primary database saving valuable CPU and I/O cycles. The

physical standby database can also be opened in read-only mode for reporting and queries.

- Performance

The Redo Apply technology used by the physical standby database applies changes using low-level recovery mechanisms, which bypass all SQL level code layers; therefore, it is the most efficient mechanism for applying high volumes of redo data.

2.1.2 Logical Standby Databases

A logical standby database is initially created as an identical copy of the primary database, but it later can be altered to have a different structure. The logical standby database is updated by executing SQL statements. This allows users to access the standby database for queries and reporting at any time. Thus, the logical standby database can be used concurrently for data protection and reporting operations.

Data Guard automatically applies information from the archived redo log file or standby redo log file to the logical standby database by transforming the data in the log files into SQL statements and then executing the SQL statements on the logical standby database. Because the logical standby database is updated using SQL statements, it must remain open. Although the logical standby database is opened in read/write mode, its target tables for the regenerated SQL are available only for read-only operations. While those tables are being updated, they can be used simultaneously for other tasks such as reporting, summations, and queries. Moreover, these tasks can be optimized by creating additional indexes and materialized views on the maintained tables.

A logical standby database has some restrictions on datatypes, types of tables, and types of DDL and DML operations. [Section 4.1.1](#) describes the unsupported datatypes and storage attributes for tables.

Benefits of a Logical Standby Database

A logical standby database provides similar disaster recovery, high availability, and data protection benefits as a physical standby database. It also provides the following specialized benefits:

- Efficient use of standby hardware resources

A logical standby database can be used for other business purposes in addition to disaster recovery requirements. It can host additional database schemas beyond the ones that are protected in a Data Guard configuration, and users can perform normal DDL or DML operations on those schemas any time. Because the logical standby tables that are protected by Data Guard can be stored in a different physical layout than on the primary database, additional indexes and materialized views can be created to improve query performance and suit specific business requirements.

- Reduction in primary database workload

A logical standby database can remain open at the same time its tables are updated from the primary database, and those tables are simultaneously available for read access. This makes a logical standby database an excellent choice to do queries, summations, and reporting activities, thereby off-loading the primary database from those tasks and saving valuable CPU and I/O cycles.

2.2 User Interfaces for Administering Data Guard Configurations

You can use the following interfaces to configure, implement, and manage a Data Guard configuration:

- Oracle Enterprise Manager
Enterprise Manager provides a GUI interface for the Data Guard broker that automates many of the tasks involved in creating, configuring, and monitoring a Data Guard environment. See *Oracle Data Guard Broker* and the Oracle Enterprise Manager online Help for information about the GUI and its wizards.
- SQL*Plus Command-line interface
Several SQL*Plus statements use the `STANDBY` keyword to specify operations on a standby database. Other SQL statements do not include standby-specific syntax, but they are useful for performing operations on a standby database. See [Chapter 15](#) for a list of the relevant statements.
- Initialization parameters
Several initialization parameters are used to define the Data Guard environment. See [Chapter 13](#) for a list of the relevant initialization parameters.
- Data Guard broker command-line interface (DGMGRL)
The DGMGRL command-line interface is an alternative to using Oracle Enterprise Manager. The DGMGRL command-line interface is useful if you want to use the broker to manage a Data Guard configuration from batch programs or scripts. See *Oracle Data Guard Broker* for complete information.

2.3 Data Guard Operational Prerequisites

The following sections describe operational requirements for using Data Guard:

- [Hardware and Operating System Requirements](#)
- [Oracle Software Requirements](#)

2.3.1 Hardware and Operating System Requirements

The following list describes hardware and operating system requirements for using Data Guard:

- All members of a Data Guard configuration must run an Oracle image that is built for the same platform.
For example, this means a Data Guard configuration with a primary database on a 32-bit Linux on Intel system can have a standby database that is configured on a 32-bit Linux on Intel system. However, a primary database on a 64-bit HP-UX system can also be configured with a standby database on a 32-bit HP-UX system, as long as both servers are running 32-bit images.
- The hardware (for example, the number of CPUs, memory size, storage configuration) can be different between the primary and standby systems.
If the standby system is smaller than the primary system, you may have to restrict the work that can be done on the standby system after a switchover or failover. The standby system must have enough resources available to receive and apply all redo data from the primary database. The logical standby database requires additional resources to translate the redo data into SQL statements and then execute the SQL on the logical standby database.

- The operating system running on the primary and standby locations must be the same, but the operating system release does not need to be the same. In addition, the standby database can use a different directory structure from the primary database.

2.3.2 Oracle Software Requirements

The following list describes Oracle software requirements for using Data Guard:

- Oracle Data Guard is available only as a feature of Oracle Database Enterprise Edition. It is not available with Oracle Database Standard Edition. This means the same release of Oracle Database Enterprise Edition must be installed on the primary database and all standby databases in a Data Guard configuration.

Note: It is possible to simulate a standby database environment with databases running Oracle Database Standard Edition. You can do this by manually transferring archived redo log files using an operating system copy utility or using custom scripts that periodically send archived redo log files from one database to the other. The consequence is that this configuration does not provide the ease-of-use, manageability, performance, and disaster-recovery capabilities available with Data Guard.

- Using Data Guard SQL Apply, you will be able to perform a rolling upgrade of the Oracle database software from patch set release n (minimally, this must be release 10.1.0.3) to the next database 10.1.0.($n+1$) patch set release. During a rolling upgrade, you can run different releases of the Oracle database on the primary and logical standby databases while you upgrade them, one at a time. For complete information, see [Chapter 11, "Using SQL Apply to Upgrade the Oracle Database"](#) and the ReadMe file for the applicable Oracle Database 10g patch set release.
- The COMPATIBLE initialization parameter must be set to the same value on all databases in a Data Guard configuration.
- If you are currently running Oracle Data Guard on Oracle8i database software, see *Oracle Database Upgrade Guide* for complete information about upgrading to Oracle Data Guard.
- The primary database must run in ARCHIVELOG mode. See *Oracle Database Administrator's Guide* for more information.
- The primary database can be a single instance database or a multi-instance Real Application Clusters database. The standby databases can be single instance databases or multi-instance Real Application Clusters (RAC) databases, and these standby databases can be a mix of both physical and logical types. See *Oracle Database High Availability Overview* for more information about configuring and using Oracle Data Guard with RAC.
- Each primary database and standby database must have its own control file.
- If a standby database is located on the same system as the primary database, the archival directories for the standby database *must* use a different directory structure than the primary database. Otherwise, the standby database may overwrite the primary database files.
- To protect against unlogged direct writes in the primary database that cannot be propagated to the standby database, turn on FORCE LOGGING at the primary

database before performing datafile backups for standby creation. Keep the database in `FORCE LOGGING` mode as long as the standby database is required.

- The user accounts you use to manage the primary and standby database instances must have `SYSDBA` system privileges.
- Oracle recommends that when you set up Oracle Automatic Storage Management (ASM) and Oracle Managed Files (OMF) in a Data Guard configuration, set it up symmetrically on the primary and standby database. That is, if any database in the Data Guard configuration uses ASM, OMF, or both, then every database in the configuration should use ASM, OMF, or both, respectively. See the scenario in [Section 12.12](#) for more information.

Note: Because some applications that perform updates involving time-based data cannot handle data entered from multiple time zones, consider setting the time zone for the primary and remote standby systems to be the same to ensure the chronological ordering of records is maintained after a role transition.

2.4 Standby Database Directory Structure Considerations

The directory structure of the various standby databases is important because it determines the path names for the standby datafiles, archived redo log files, and standby redo log files. If possible, the datafiles, log files, and control files on the primary and standby systems should have the same names and path names and use Optimal Flexible Architecture (OFA) naming conventions. The archival directories on the standby database should also be identical between sites, including size and structure. This strategy allows other operations such as backups, switchovers, and failovers to execute the same set of steps, reducing the maintenance complexity.

Otherwise, you must set the filename conversion parameters (as shown in [Table 2-1](#)) or rename the datafile. Nevertheless, if you need to use a system with a different directory structure or place the standby and primary databases on the same system, you can do so with a minimum of extra administration.

The three basic configuration options are illustrated in [Figure 2-1](#). These include:

- A standby database on the same system as the primary database that uses a different directory structure than the primary system. This is illustrated in [Figure 2-1](#) as `Standby1`.

If you have a standby database on the same system as the primary database, you *must* use a different directory structure. Otherwise, the standby database attempts to overwrite the primary database files.

- A standby database on a separate system that uses the same directory structure as the primary system. This is illustrated in [Figure 2-1](#) as `Standby2`. This is the recommended method.
- A standby database on a separate system that uses a different directory structure than the primary system. This is illustrated in [Figure 2-1](#) as `Standby3`.

Note: if any database in the Data Guard configuration uses ASM, OMF, or both, then every database in the configuration should use ASM, OMF, or both, respectively. See [Chapter 12](#) for a scenario describing how to set up OMF in a Data Guard configuration.

Figure 2–1 Possible Standby Configurations

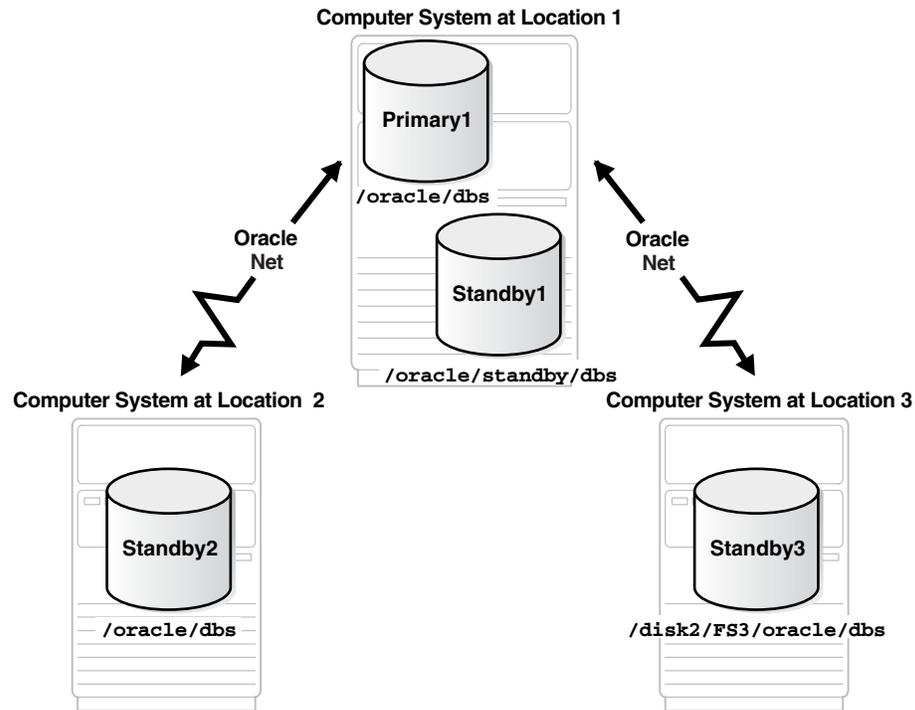


Table 2–1 describes possible configurations of primary and standby databases and the consequences of each. In the table, note that the service name defaults to the concatenation of the `DB_UNIQUE_NAME` and `DB_DOMAIN` initialization parameters. You must specify a unique value for the `DB_UNIQUE_NAME` initialization parameter when more than one member of a Data Guard configuration resides on the same system. Oracle recommends that the value of the `DB_UNIQUE_NAME` initialization parameter always be unique, even if each database is located on a separate system.

Table 2–1 Standby Database Location and Directory Options

Standby System	Directory Structure	Consequences
Same as primary system	Different than primary system (required)	<ul style="list-style-type: none"> You must set the <code>DB_UNIQUE_NAME</code> initialization parameter. You can either manually rename files or set up the <code>DB_FILE_NAME_CONVERT</code> and <code>LOG_FILE_NAME_CONVERT</code> initialization parameters on the standby database to automatically update the path names for primary database datafiles and archived redo log files and standby redo log files in the standby database control file. (See Section 3.1.4.) The standby database does not protect against disasters that destroy the system on which the primary and standby databases reside, but it does provide switchover capabilities for planned maintenance.

Table 2–1 (Cont.) Standby Database Location and Directory Options

Standby System	Directory Structure	Consequences
Separate system	Same as primary system	<ul style="list-style-type: none"> ■ You do not need to rename primary database files, archived redo log files, and standby redo log files in the standby database control file, although you can still do so if you want a new naming scheme (for example, to spread the files among different disks). ■ By locating the standby database on separate physical media, you safeguard the data on the primary database against disasters that destroy the primary system.
Separate system	Different than primary system	<ul style="list-style-type: none"> ■ You can either manually rename files or set up the <code>DB_FILE_NAME_CONVERT</code> and <code>LOG_FILE_NAME_CONVERT</code> initialization parameters on the standby database to automatically rename the datafiles (see Section 3.1.4). ■ By locating the standby database on separate physical media, you safeguard the data on the primary database against disasters that destroy the primary system.

2.5 Online Redo Logs, Archived Redo Logs, and Standby Redo Logs

The most crucial structures for Data Guard recovery operations are online redo logs, archived redo logs, and standby redo logs. Redo data transmitted from the primary database is received by the remote file server (RFS) process on the standby system where the RFS process writes the redo data to archived log files or standby redo log files. Redo data can be applied either after the redo is written to the archived redo log file or standby redo log file, or, if real-time apply is enabled, directly from the standby redo log file as it is being filled.

This documentation assumes that you already understand the concepts behind online redo logs and archived redo logs. [Section 2.5.1](#) supplements the basic concepts by providing information that is specific to Data Guard configurations. [Section 2.5.2](#) provides detailed information about using standby redo log files.

See *Oracle Database Administrator's Guide* for more information about redo logs and archive logs, and [Section 6.2.1](#) for information about real-time apply.

2.5.1 Online Redo Logs and Archived Redo Logs

The transmission of redo is integral to maintaining the transactional consistency of the primary and standby databases. Both online redo logs and archived redo logs are required in a Data Guard environment:

- Online redo logs

Every instance of an Oracle primary database and logical standby database has an online redo log to protect the database in case of an instance failure. Physical standby databases do not use an online redo log, because physical standby databases are not opened for read/write I/O. Changes are not made to the physical standby database and new redo data is not generated.
- Archived redo logs

An archived redo log is required because archiving is the method used to keep standby databases transactionally consistent with the primary database. Primary databases, and both physical and logical standby databases all use an archived redo log. Oracle databases are set up, by default, to run in ARCHIVELOG mode so

that the archiver (ARC*n*) process automatically copies each filled online redo log file to one or more archived redo log files.

Unlike physical standby databases, logical standby databases are open databases that generate redo data and have multiple log files, including online redo log files, archived redo log files, and standby redo log files (if configured).

Both the size of the online redo log files and the frequency with which a log switch occurs can affect the generation of the archived redo log files at the primary site. The *Oracle Database High Availability Overview* provides recommendations for log group sizing.

An Oracle database will attempt a checkpoint at each log switch. Therefore, if the size of the online redo log file is too small, frequent log switches lead to frequent checkpointing and negatively affect system performance on the standby database.

See Also: *Oracle Database Administrator's Guide* for more details about configuring redo logs, archive logs, and log groups

2.5.2 Standby Redo Logs

A standby redo log is similar to an online redo log, except that a standby redo log is used to store redo data received from another database.

A standby redo log is required if you want to implement:

- The maximum protection and maximum availability levels of data protection (described in [Section 1.4](#) and in more detail in [Section 5.6](#))
- Real-time apply (described in [Section 6.2](#))
- Cascaded destinations (described in [Appendix E](#))

A standby redo log provides a number of advantages:

- Standby redo log files can reside on raw devices, which may be important if either or both the primary and standby databases reside in a Real Application Clusters environment.
- Standby redo log files can be multiplexed using multiple members, improving reliability over archived log files.
- During a failover, Data Guard can recover and apply more redo data from standby redo log files than from the archived log files alone.
- The archiver (ARC*n*) process or the log writer (LGWR) process on the primary database can transmit redo data directly to remote standby redo log files, potentially eliminating the need to register a partial archived log file (for example, to recover after a standby database crashes). See [Chapter 5](#) for more information.

[Section 3.1.3](#) describes how to configure standby redo log files.

Creating a Physical Standby Database

This chapter steps you through the process of creating a physical standby database. It includes the following main topics:

- [Preparing the Primary Database for Standby Database Creation](#)
- [Step-by-Step Instructions for Creating a Physical Standby Database](#)
- [Post-Creation Steps](#)

The steps described in this chapter configure the standby database for maximum performance mode, which is the default data protection mode. [Chapter 5](#) provides information about configuring the different data protection modes. The discussions in this chapter assume that you specify initialization parameters in a server parameter file (SPFILE), instead of a text initialization parameter file (PFILE).

See also:

- *Oracle Database Administrator's Guide* for information about creating and using server parameter files
- *Oracle Data Guard Broker* and the Enterprise Manager online help system for information about using the graphical user interface to automatically create a physical standby database

3.1 Preparing the Primary Database for Standby Database Creation

Before you create a standby database you must first ensure the primary database is properly configured.

[Table 3–1](#) provides a checklist of the tasks that you perform on the primary database to prepare for physical standby database creation. There is also a reference to the section that describes the task in more detail.

Table 3–1 *Preparing the Primary Database for Physical Standby Database Creation*

Reference	Task
Section 3.1.1	Enable Forced Logging
Section 3.1.2	Create a Password File
Section 3.1.3	Configure a Standby Redo Log
Section 3.1.4	Set Primary Database Initialization Parameters
Section 3.1.5	Enable Archiving

Note: Perform these preparatory tasks only once. After you complete these steps, the database is prepared to serve as the primary database for one or more standby databases.

3.1.1 Enable Forced Logging

Place the primary database in `FORCE LOGGING` mode after database creation using the following SQL statement:

```
SQL> ALTER DATABASE FORCE LOGGING;
```

This statement can take a considerable amount of time to complete, because it waits for all unlogged direct write I/O to finish.

3.1.2 Create a Password File

Create a password file if one does not already exist. Every database in a Data Guard configuration must use a password file, and the password for the `SYS` user must be identical on every system for redo data transmission to succeed. See *Oracle Database Administrator's Guide*.

3.1.3 Configure a Standby Redo Log

A standby redo log is required for the maximum protection and maximum availability modes and the `LGWR ASYNC` transport mode is recommended for all databases. Data Guard can recover and apply more redo data from a standby redo log than from archived redo log files alone.

You should plan the standby redo log configuration and create all required log groups and group members when you create the standby database. For increased availability, consider multiplexing the standby redo log files, similar to the way that online redo log files are multiplexed.

Perform the following steps to configure the standby redo log.

Step 1 Ensure log file sizes are identical on the primary and standby databases.

The size of the current standby redo log files must exactly match the size of the current primary database online redo log files. For example, if the primary database uses two online redo log groups whose log files are 200K, then the standby redo log groups should also have log file sizes of 200K.

Step 2 Determine the appropriate number of standby redo log file groups.

Minimally, the configuration should have one more standby redo log file group than the number of online redo log file groups on the primary database. However, the *recommended* number of standby redo log file groups is dependent on the number of threads on the primary database. Use the following equation to determine an appropriate number of standby redo log file groups:

$(\text{maximum number of logfiles for each thread} + 1) * \text{maximum number of threads}$

Using this equation reduces the likelihood that the primary instance's log writer (LGWR) process will be blocked because a standby redo log file cannot be allocated on the standby database. For example, if the primary database has 2 log files for each thread and 2 threads, then 6 standby redo log file groups are needed on the standby database.

Note: Logical standby databases may require more standby redo log files (or additional ARC*n* processes) depending on the workload. This is because logical standby databases also write to online redo log files, which take precedence over standby redo log files. Thus, the standby redo log files may not be archived as quickly as the online redo log files. Also, see [Section 5.7.3.1](#).

Step 3 Verify related database parameters and settings.

Verify the values used for the MAXLOGFILES and MAXLOGMEMBERS clauses on the SQL CREATE DATABASE statement will not limit the number of standby redo log file groups and members that you can add. The only way to override the limits specified by the MAXLOGFILES and MAXLOGMEMBERS clauses is to re-create the primary database or control file.

See *Oracle Database SQL Reference* and your operating system specific Oracle documentation for the default and legal values of the MAXLOGFILES and MAXLOGMEMBERS clauses.

Step 4 Create standby redo log file groups.

To create new standby redo log file groups and members, you must have the ALTER DATABASE system privilege. The standby database begins using the newly created standby redo data the next time there is a log switch on the primary database. [Example 3-1](#) and [Example 3-2](#) show how to create a new standby redo log file group using the ALTER DATABASE statement with variations of the ADD STANDBY LOGFILE GROUP clause.

Example 3-1 Adding a Standby Redo Log File Group to a Specific Thread

The following statement adds a new standby redo log file group to a standby database and assigns it to THREAD 5:

```
SQL> ALTER DATABASE ADD STANDBY LOGFILE THREAD 5
  2> ('/oracle/dbs/log1c.rdo', '/oracle/dbs/log2c.rdo') SIZE 500M;
```

The THREAD clause is required only if you want to add one or more standby redo log file groups to a *specific* primary database thread. If you do not include the THREAD clause and the configuration uses Real Application Clusters (RAC), Data Guard will automatically assign standby redo log file groups to threads at runtime as they are needed by the various RAC instances.

Example 3-2 Adding a Standby Redo Log File Group to a Specific Group Number

You can also specify a number that identifies the group using the GROUP clause:

```
SQL> ALTER DATABASE ADD STANDBY LOGFILE GROUP 10
  2> ('/oracle/dbs/log1c.rdo', '/oracle/dbs/log2c.rdo') SIZE 500M;
```

Using group numbers can make administering standby redo log file groups easier. However, the group number must be between 1 and the value of the MAXLOGFILES clause. Do not skip log file group numbers (that is, do not number groups 10, 20, 30, and so on), or you will use additional space in the standby database control file.

Note: Although the standby redo log is only used when the database is running in the standby role, Oracle recommends that you create a standby redo log on the primary database so that the primary database can switch over quickly to the standby role without the need for additional DBA intervention. Consider using Oracle Enterprise Manager to automatically configure standby redo log on both your primary and standby databases.

Step 5 Verify the standby redo log file groups were created.

To verify the standby redo log file groups are created and running correctly, invoke a log switch on the primary database, and then query either the V\$STANDBY_LOG view or the V\$LOGFILE view on the standby database once it has been created. For example:

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

GROUP#	THREAD#	SEQUENCE#	ARC	STATUS
3	1	16	NO	ACTIVE
4	0	0	YES	UNASSIGNED
5	0	0	YES	UNASSIGNED

3.1.4 Set Primary Database Initialization Parameters

On the primary database, you define initialization parameters that control redo transport services while the database is in the primary role. There are additional parameters you need to add that control the receipt of the redo data and log apply services when the primary database is transitioned to the standby role.

[Example 3-3](#) shows the primary role initialization parameters that you maintain on the primary database. This example represents a Data Guard configuration with a primary database located in Chicago and one physical standby database located in Boston. The parameters shown in [Example 3-3](#) are valid for the Chicago database when it is running in either the primary or the standby database role. The configuration examples use the names shown in the following table:

Database	DB_UNIQUE_NAME	Oracle Net Service Name
Primary	chicago	chicago
Physical standby	boston	boston

Example 3-3 Primary Database: Primary Role Initialization Parameters

```
DB_NAME=chicago
DB_UNIQUE_NAME=chicago
LOG_ARCHIVE_CONFIG='DG_CONFIG=(chicago,boston) '
CONTROL_FILES='/arch1/chicago/control1.ct1', '/arch2/chicago/control2.ct1'
LOG_ARCHIVE_DEST_1=
  'LOCATION=/arch1/chicago/
  VALID_FOR=(ALL_LOGFILES,ALL_ROLES)
  DB_UNIQUE_NAME=chicago'
LOG_ARCHIVE_DEST_2=
  'SERVICE=boston LGWR ASYNC
  VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)
  DB_UNIQUE_NAME=boston'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
```

```
LOG_ARCHIVE_DEST_STATE_2=ENABLE
REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE
LOG_ARCHIVE_FORMAT=%t_%s_%r.arc
LOG_ARCHIVE_MAX_PROCESSES=30
```

These parameters control how redo transport services transmit redo data to the standby system and the archiving of redo data on the local file system. Note that the example specifies the LGWR process and asynchronous (ASYNC) network transmission to transmit redo data on the LOG_ARCHIVE_DEST_2 initialization parameter. These are the recommended settings and require standby redo log files (see [Section 3.1.3, "Configure a Standby Redo Log"](#) on page 3-2).

[Example 3-4](#) shows the additional standby role initialization parameters on the primary database. These parameters take effect when the primary database is transitioned to the standby role.

Example 3-4 Primary Database: Standby Role Initialization Parameters

```
FAL_SERVER=boston
FAL_CLIENT=chicago
DB_FILE_NAME_CONVERT='boston','chicago'
LOG_FILE_NAME_CONVERT=
'/arch1/boston','/arch1/chicago','/arch2/boston','/arch2/chicago/'
STANDBY_FILE_MANAGEMENT=AUTO
```

Specifying the initialization parameters shown in [Example 3-4](#) sets up the primary database to resolve gaps, converts new datafile and log file path names from a new primary database, and archives the incoming redo data when this database is in the standby role. With the initialization parameters for both the primary and standby roles set as described, none of the parameters need to change after a role transition.

The following table provides a brief explanation about each parameter setting shown in [Example 3-3](#) and [Example 3-4](#).

Parameter	Recommended Setting
DB_NAME	Specify an 8-character name. Use the same name for all standby databases.
DB_UNIQUE_NAME	Specify a unique name for each database. This name stays with the database and does not change, even if the primary and standby databases reverse roles.
LOG_ARCHIVE_CONFIG	Specify the DG_CONFIG attribute on this parameter to list the DB_UNIQUE_NAME of the primary and standby databases in the Data Guard configuration; this enables the dynamic addition of a standby database to a Data Guard configuration that has a Real Application Clusters primary database running in either maximum protection or maximum availability mode. By default, the LOG_ARCHIVE_CONFIG parameter enables the database to send and receive redo; after a role transition, you may need to specify these settings again using the SEND, NOSEND, RECEIVE, or NORECEIVE keywords.
CONTROL_FILES	Specify the path name for the control files on the primary database. Example 3-3 shows how to do this for two control files. It is recommended that a second copy of the control file is available so an instance can be easily restarted after copying the good control file to the location of the bad control file.

Parameter	Recommended Setting
LOG_ARCHIVE_DEST_n	<p>Specify where the redo data is to be archived on the primary and standby systems. In Example 3-3:</p> <ul style="list-style-type: none"> ■ LOG_ARCHIVE_DEST_1 archives redo data generated by the primary database from the local online redo log files to the local archived redo log files in /arch1/chicago/. ■ LOG_ARCHIVE_DEST_2 is valid only for the primary role. This destination transmits redo data to the remote physical standby destination boston. <p>Note: If a flash recovery area was configured (with the DB_RECOVERY_FILE_DEST initialization parameter) and you have not explicitly configured a local archiving destination with the LOCATION attribute, Data Guard automatically uses the LOG_ARCHIVE_DEST_10 initialization parameter as the default destination for local archiving. See Section 5.2.3 for more information. Also, see Chapter 14 for complete LOG_ARCHIVE_DEST_n information.</p>
LOG_ARCHIVE_DEST_STATE_n	Specify ENABLE to allow redo transport services to transmit redo data to the specified destination.
REMOTE_LOGIN_PASSWORDFILE	Set the same password for SYS on both the primary and standby databases. The recommended setting is either EXCLUSIVE or SHARED.
LOG_ARCHIVE_FORMAT	Specify the format for the archived redo log files using a thread (%t), sequence number (%s), and resetlogs ID (%r). See Section 5.7.1 for another example.
LOG_ARCHIVE_MAX_PROCESSES =integer	Specify the maximum number (from 1 to 30) of archiver (ARCn) processes you want Oracle software to invoke initially. The default value is 4. See Section 5.3.1.2 for more information about ARCn processing.
FAL_SERVER	Specify the Oracle Net service name of the FAL server (typically this is the database running in the primary role). When the Chicago database is running in the standby role, it uses the Boston database as the FAL server from which to fetch (request) missing archived redo log files if Boston is unable to automatically send the missing log files. See Section 5.8 .
FAL_CLIENT	Specify the Oracle Net service name of the Chicago database. The FAL server (Boston) copies missing archived redo log files to the Chicago standby database. See Section 5.8 .
DB_FILE_NAME_CONVERT	Specify the path name and filename location of the primary database datafiles followed by the standby location. This parameter converts the path names of the primary database datafiles to the standby datafile path names. If the standby database is on the same system as the primary database or if the directory structure where the datafiles are located on the standby site is different from the primary site, then this parameter is required. Note that this parameter is used only to convert path names for physical standby databases. Multiple pairs of paths may be specified by this parameter.
LOG_FILE_NAME_CONVERT	Specify the location of the primary database online redo log files followed by the standby location. This parameter converts the path names of the primary database log files to the path names on the standby database. If the standby database is on the same system as the primary database or if the directory structure where the log files are located on the standby system is different from the primary system, then this parameter is required. Multiple pairs of paths may be specified by this parameter.
STANDBY_FILE_MANAGEMENT	Set to AUTO so when datafiles are added to or dropped from the primary database, corresponding changes are made automatically to the standby database.

Caution: Review the initialization parameter file for additional parameters that may need to be modified. For example, you may need to modify the dump destination parameters (`BACKGROUND_DUMP_DEST`, `CORE_DUMP_DEST`, `USER_DUMP_DEST`) if the directory location on the standby database is different from those specified on the primary database. In addition, you may have to create directories on the standby system if they do not already exist.

3.1.5 Enable Archiving

If archiving is not enabled, issue the following statements to put the primary database in ARCHIVELOG mode and enable automatic archiving:

```
SQL> SHUTDOWN IMMEDIATE;
SQL> STARTUP MOUNT;
SQL> ALTER DATABASE ARCHIVELOG;
SQL> ALTER DATABASE OPEN;
```

See *Oracle Database Administrator's Guide* for information about archiving.

3.2 Step-by-Step Instructions for Creating a Physical Standby Database

This section describes the tasks you perform to create a physical standby database.

[Table 3–2](#) provides a checklist of the tasks that you perform to create a physical standby database and the database or databases on which you perform each task. There is also a reference to the section that describes the task in more detail.

Table 3–2 *Creating a Physical Standby Database*

Reference	Task	Database
Section 3.2.1	Create a Backup Copy of the Primary Database Datafiles	Primary
Section 3.2.2	Create a Control File for the Standby Database	Primary
Section 3.2.3	Prepare an Initialization Parameter File for the Standby Database	Primary
Section 3.2.4	Copy Files from the Primary System to the Standby System	Primary
Section 3.2.5	Set Up the Environment to Support the Standby Database	Standby
Section 3.2.6	Start the Physical Standby Database	Standby
Section 3.2.7	Verify the Physical Standby Database Is Performing Properly	Standby

3.2.1 Create a Backup Copy of the Primary Database Datafiles

You can use any backup copy of the primary database to create the physical standby database, as long as you have the necessary archived redo log files to completely recover the database. Oracle recommends that you use the Recovery Manager utility (RMAN).

See *Oracle High Availability Architecture and Best Practices* for backup recommendations and *Oracle Database Backup and Recovery Advanced User's Guide* to perform an RMAN backup operation.

3.2.2 Create a Control File for the Standby Database

If the backup procedure required you to shut down the primary database, issue the following SQL*Plus statement to start the primary database:

```
SQL> STARTUP MOUNT;
```

Then, create the control file for the standby database, and open the primary database to user access, as shown in the following example:

```
SQL> ALTER DATABASE CREATE STANDBY CONTROLFILE AS '/tmp/boston.ctl';
SQL> ALTER DATABASE OPEN;
```

Note: You cannot use a single control file for both the primary and standby databases.

3.2.3 Prepare an Initialization Parameter File for the Standby Database

Perform the following steps to create a standby initialization parameter file.

Step 1 Copy the primary database parameter file to the standby database.

Create a text initialization parameter file (PFILE) from the server parameter file (SPFILE) used by the primary database; a text initialization parameter file can be copied to the standby location and modified. For example:

```
SQL> CREATE PFILE='/tmp/initboston.ora' FROM SPFILE;
```

Later, in [Section 3.2.5](#), you will convert this file back to a server parameter file after it is modified to contain the parameter values appropriate for use with the physical standby database.

Step 2 Set initialization parameters on the physical standby database.

Although most of the initialization parameter settings in the text initialization parameter file that you copied from the primary system are also appropriate for the physical standby database, some modifications need to be made.

[Example 3–5](#) shows the portion of the standby initialization parameter file where values were modified for the physical standby database. Parameter values that are different from [Example 3–3](#) and [Example 3–4](#) are shown in bold typeface. The parameters shown in [Example 3–5](#) are valid for the Boston database when it is running in either the primary or the standby database role.

Example 3–5 Modifying Initialization Parameters for a Physical Standby Database

```
.
.
.
DB_NAME=chicago
DB_UNIQUE_NAME=boston
LOG_ARCHIVE_CONFIG=(chicago,boston)
CONTROL_FILES='/arch1/boston/control1.ctl', '/arch2/boston/control2.ctl'
DB_FILE_NAME_CONVERT='chicago','boston'
LOG_FILE_NAME_CONVERT=
  '/arch1/chicago/', '/arch1/boston/', '/arch2/chicago/', '/arch2/boston/'
LOG_ARCHIVE_FORMAT=log%t_%s_%r.arc
LOG_ARCHIVE_DEST_1=
  'LOCATION=/arch1/boston/
    VALID_FOR=(ALL_LOGFILES,ALL_ROLES)
```

```

DB_UNIQUE_NAME=boston'
LOG_ARCHIVE_DEST_2=
'SERVICE=chicago LGWR ASYNC
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)
DB_UNIQUE_NAME=chicago'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_STATE_2=ENABLE
REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE
STANDBY_FILE_MANAGEMENT=AUTO
FAL_SERVER=chicago
FAL_CLIENT=boston
.
.
.

```

Note that the example assumes the use of the LGWR process to transmit redo data to both the local and remote destinations on the LOG_ARCHIVE_DEST_2 initialization parameter.

In addition, ensure the COMPATIBLE initialization parameter is set to the same value on both the primary and standby databases. If the values differ, redo transport services may be unable to transmit redo data from the primary database to the standby databases. In a Data Guard configuration, COMPATIBLE must be set to a minimum of 9.2.0.1.0. However, if you want to take advantage of new Oracle Database 10g features, set the COMPATIBLE parameter to 10.2.0.0 or higher.

It is always a good practice to use the SHOW PARAMETERS command to verify no other parameters need to be changed.

The following table provides a brief explanation about the parameter settings shown in [Example 3–5](#) that have different settings from the primary database.

Parameter	Recommended Setting
DB_UNIQUE_NAME	Specify a unique name for this database. This name stays with the database and does not change even if the primary and standby databases reverse roles.
CONTROL_FILES	Specify the path name for the control files on the standby database. Example 3–5 shows how to do this for two control files. It is recommended that a second copy of the control file is available so an instance can be easily restarted after copying the good control file to the location of the bad control file.
DB_FILE_NAME_CONVERT	Specify the path name and filename location of the primary database datafiles followed by the standby location. This parameter converts the path names of the primary database datafiles to the standby datafile path names. If the standby database is on the same system as the primary database or if the directory structure where the datafiles are located on the standby site is different from the primary site, then this parameter is required.
LOG_FILE_NAME_CONVERT	Specify the location of the primary database online redo log files followed by the standby location. This parameter converts the path names of the primary database log files to the path names on the standby database. If the standby database is on the same system as the primary database or if the directory structure where the log files are located on the standby system is different from the primary system, then this parameter is required.

Parameter	Recommended Setting
LOG_ARCHIVE_DEST_ <i>n</i>	<p>Specify where the redo data is to be archived. In Example 3-5:</p> <ul style="list-style-type: none"> LOG_ARCHIVE_DEST_1 archives redo data received from the primary database to archived redo log files in /arch1/boston/. LOG_ARCHIVE_DEST_2 is currently ignored because this destination is valid only for the primary role. If a switchover occurs and this instance becomes the primary database, then it will transmit redo data to the remote Chicago destination. <p>Note: If a flash recovery area was configured (with the DB_RECOVERY_FILE_DEST initialization parameter) and you have not explicitly configured a local archiving destination with the LOCATION attribute, Data Guard automatically uses the LOG_ARCHIVE_DEST_10 initialization parameter as the default destination for local archiving. See Section 5.2.3 for more information. Also, see Chapter 14 for complete information about LOG_ARCHIVE_DEST_<i>n</i>.</p>
FAL_SERVER	<p>Specify the Oracle Net service name of the FAL server (typically this is the database running in the primary role). When the Boston database is running in the standby role, it uses the Chicago database as the FAL server from which to fetch (request) missing archived redo log files if Chicago is unable to automatically send the missing log files. See Section 5.8.</p>
FAL_CLIENT	<p>Specify the Oracle Net service name of the Boston database. The FAL server (Chicago) copies missing archived redo log files to the Boston standby database. See Section 5.8.</p>

Caution: Review the initialization parameter file for additional parameters that may need to be modified. For example, you may need to modify the dump destination parameters (BACKGROUND_DUMP_DEST, CORE_DUMP_DEST, USER_DUMP_DEST) if the directory location on the standby database is different from those specified on the primary database. In addition, you may have to create directories on the standby system if they do not already exist.

3.2.4 Copy Files from the Primary System to the Standby System

Use an operating system copy utility to copy the following binary files from the primary system to the standby system:

- Backup datafiles created in [Section 3.2.1](#)
- Standby control file created in [Section 3.2.2](#)
- Initialization parameter file created in [Section 3.2.3](#)

3.2.5 Set Up the Environment to Support the Standby Database

Perform the following steps to create a Windows-based service, create a password file, set up the Oracle Net environment, and create a SPFILE.

Step 1 Create a Windows-based service.

If the standby system is running on a Windows-based system, use the ORADIM utility to create a Windows Service and password file. For example:

```
WINNT> oradim -NEW -SID boston -INTPWD password -STARTMODE manual
```

See *Oracle Database Platform Guide for Microsoft Windows (32-Bit)* for more information about using the ORADIM utility.

Step 2 Create a password file.

On platforms other than Windows, create a password file, and set the password for the SYS user to the same password used by the SYS user on the primary database. The password for the SYS user on every database in a Data Guard configuration must be identical for redo transmission to succeed. See *Oracle Database Administrator's Guide*.

Step 3 Configure listeners for the primary and standby databases.

On both the primary and standby sites, use Oracle Net Manager to configure a listener for the respective databases.

To restart the listeners (to pick up the new definitions), enter the following LSNRCTL utility commands on both the primary and standby systems:

```
% lsnrctl stop
% lsnrctl start
```

See *Oracle Database Net Services Administrator's Guide*.

Step 4 Create Oracle Net service names.

On both the primary and standby systems, use Oracle Net Manager to create a network service name for the primary and standby databases that will be used by redo transport services.

The Oracle Net service name must resolve to a connect descriptor that uses the same protocol, host address, port, and service that you specified when you configured the listeners for the primary and standby databases. The connect descriptor must also specify that a dedicated server be used.

See the *Oracle Database Net Services Administrator's Guide* and the *Oracle Database Administrator's Guide*.

Step 5 Create a server parameter file for the standby database.

On an idle standby database, use the SQL CREATE statement to create a server parameter file for the standby database from the text initialization parameter file that was edited in Step 2 on page 3-8. For example:

```
SQL> CREATE SPFILE FROM PFILE='initboston.ora';
```

3.2.6 Start the Physical Standby Database

Perform the following steps to start the physical standby database and Redo Apply.

Step 1 Start the physical standby database.

On the standby database, issue the following SQL statement to start and mount the database:

```
SQL> STARTUP MOUNT;
```

Step 2 Start Redo Apply.

On the standby database, issue the following command to start Redo Apply:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE DISCONNECT FROM SESSION;
```

The statement includes the DISCONNECT FROM SESSION option so that Redo Apply runs in a background session. See [Section 6.3, "Applying Redo Data to Physical Standby Databases"](#) for more information.

Step 3 Test archival operations to the physical standby database.

In this example, the transmission of redo data to the remote standby location does not occur until after a log switch. A log switch occurs, by default, when an online redo log file becomes full. To force a log switch so that redo data is transmitted immediately, use the following `ALTER SYSTEM SWITCH LOGFILE;` statement on the primary database. For example:

```
SQL> ALTER SYSTEM SWITCH LOGFILE;
```

3.2.7 Verify the Physical Standby Database Is Performing Properly

Once you create the physical standby database and set up redo transport services, you may want to verify database modifications are being successfully transmitted from the primary database to the standby database.

To see that redo data is being received on the standby database, you should first identify the existing archived redo log files on the standby database, force a log switch and archive a few online redo log files on the primary database, and then check the standby database again. The following steps show how to perform these tasks.

Step 1 Identify the existing archived redo log files.

On the standby database, query the `V$ARCHIVED_LOG` view to identify existing files in the archived redo log. For example:

```
SQL> SELECT SEQUENCE#, FIRST_TIME, NEXT_TIME
       2 FROM V$ARCHIVED_LOG ORDER BY SEQUENCE#;
```

SEQUENCE#	FIRST_TIME	NEXT_TIME
8	11-JUL-02 17:50:45	11-JUL-02 17:50:53
9	11-JUL-02 17:50:53	11-JUL-02 17:50:58
10	11-JUL-02 17:50:58	11-JUL-02 17:51:03

3 rows selected.

Step 2 Force a log switch to archive the current online redo log file.

On the primary database, issue the `ALTER SYSTEM SWITCH LOGFILE` statement to force a log switch and archive the current online redo log file group:

```
SQL> ALTER SYSTEM SWITCH LOGFILE;
```

Step 3 Verify the new redo data was archived on the standby database.

On the standby database, query the `V$ARCHIVED_LOG` view to verify the redo data was received and archived on the standby database:

```
SQL> SELECT SEQUENCE#, FIRST_TIME, NEXT_TIME
       2> FROM V$ARCHIVED_LOG ORDER BY SEQUENCE#;
```

SEQUENCE#	FIRST_TIME	NEXT_TIME
8	11-JUL-02 17:50:45	11-JUL-02 17:50:53
9	11-JUL-02 17:50:53	11-JUL-02 17:50:58
10	11-JUL-02 17:50:58	11-JUL-02 17:51:03
11	11-JUL-02 17:51:03	11-JUL-02 18:34:11

4 rows selected.

The archived redo log files are now available to be applied to the physical standby database.

Step 4 Verify new archived redo log files were applied.

On the standby database, query the V\$ARCHIVED_LOG view to verify the archived redo log files were applied.

```
SQL> SELECT SEQUENCE#,APPLIED FROM V$ARCHIVED_LOG
       2 ORDER BY SEQUENCE#;
```

```
SEQUENCE# APP
----- ---
          8 YES
          9 YES
         10 YES
         11 YES
```

4 rows selected.

See [Section 5.9.1, "Monitoring Log File Archival Information"](#) and [Section 8.5.4, "Monitoring Log Apply Services on Physical Standby Databases"](#) to verify redo transport services and log apply services are working correctly.

3.3 Post-Creation Steps

At this point, the physical standby database is running and can provide the maximum performance level of data protection. The following list describes additional preparations you can take on the physical standby database:

- Upgrade the data protection mode

The Data Guard configuration is initially set up in the maximum performance mode (the default). See [Section 5.6](#) for information about the data protection modes and how to upgrade or downgrade the current protection mode.

- Enable Flashback Database

Flashback Database removes the need to re-create the primary database after a failover. Flashback Database enables you to return a database to its state at a time in the recent past much faster than traditional point-in-time recovery, because it does not require restoring datafiles from backup nor the extensive application of redo data. You can enable Flashback Database on the primary database, the standby database, or both. See [Section 12.4](#) and [Section 12.5](#) for scenarios showing how to use Flashback Database in a Data Guard environment. Also, see *Oracle Database Backup and Recovery Advanced User's Guide* for more information about Flashback Database.

Creating a Logical Standby Database

This chapter steps you through the process of creating a logical standby database. It includes the following main topics:

- [Prerequisite Conditions for Creating a Logical Standby Database](#)
- [Step-by-Step Instructions for Creating a Logical Standby Database](#)
- [Post-Creation Steps](#)

See Also:

- *Oracle Database Administrator's Guide* for information about creating and using server parameter files
- *Oracle Data Guard Broker* and the Oracle Enterprise Manager online help system for information about using the graphical user interface to automatically create a logical standby database

4.1 Prerequisite Conditions for Creating a Logical Standby Database

Before you create a logical standby database, you must first ensure the primary database is properly configured. [Table 4-1](#) provides a checklist of the tasks that you perform on the primary database to prepare for logical standby database creation. There is also a reference to the section that describes the task in more detail.

Table 4-1 *Preparing the Primary Database for Logical Standby Database Creation*

Reference	Task
Section 4.1.1	Determine Support for Data Types and Storage Attributes for Tables
Section 4.1.2	Ensure Table Rows in the Primary Database Can Be Uniquely Identified

4.1.1 Determine Support for Data Types and Storage Attributes for Tables

Before setting up a logical standby database, ensure the logical standby database can maintain the data types and tables in your primary database. See [Appendix C](#) for a complete list of data type and storage type considerations.

4.1.2 Ensure Table Rows in the Primary Database Can Be Uniquely Identified

The physical organization in a logical standby database is different from that of the primary database, even though the logical standby database is created from a backup copy of the primary database. Thus, ROWIDs contained in the redo records generated by the primary database cannot be used to identify the corresponding row in the logical standby database.

Oracle uses primary-key or unique-constraint/index supplemental logging to logically identify a modified row in the logical standby database. When database-wide primary-key and unique-constraint/index supplemental logging is enabled, each UPDATE statement also writes the column values necessary in the redo log to uniquely identify the modified row in the logical standby database.

- If a table has a primary key defined, then the primary key is logged along with the modified columns as part of the UPDATE statement to identify the modified row.
- In the absence of a primary key, the shortest nonnull unique-constraint/index is logged along with the modified columns as part of the UPDATE statement to identify the modified row.
- In the absence of both a primary key and a nonnull unique constraint/index, all columns of bounded size are logged as part of the UPDATE statement to identify the modified row. In other words, all columns except those with the following types are logged: LONG, LOB, LONG RAW, object type, and collections.

Oracle recommends that you add a primary key or a nonnull unique index to tables in the primary database, whenever possible, to ensure that SQL Apply can efficiently apply redo data updates to the logical standby database.

Perform the following steps to ensure SQL Apply can uniquely identify rows of each table being replicated in the logical standby database.

Step 1 Find tables without unique logical identifier in the primary database.

Query the DBA_LOGSTDBY_NOT_UNIQUE view to display a list of tables that SQL Apply may not be able to uniquely identify. For example:

```
SQL> SELECT OWNER, TABLE_NAME FROM DBA_LOGSTDBY_NOT_UNIQUE
 2> WHERE (OWNER, TABLE_NAME) NOT IN
 3> (SELECT DISTINCT OWNER, TABLE_NAME FROM DBA_LOGSTDBY_UNSUPPORTED)
 4> AND BAD_COLUMN = 'Y'
```

Step 2 Add a disabled primary-key RELY constraint.

If your application ensures the rows in a table are unique, you can create a disabled primary key RELY constraint on the table. This avoids the overhead of maintaining a primary key on the primary database.

To create a disabled RELY constraint on a primary database table, use the ALTER TABLE statement with a RELY DISABLE clause. The following example creates a disabled RELY constraint on a table named mytab, for which rows can be uniquely identified using the id and name columns:

```
SQL> ALTER TABLE mytab ADD PRIMARY KEY (id, name) RELY DISABLE;
```

When you specify the RELY constraint, the system will assume that rows are unique. Because you are telling the system to rely on the information, but are not validating it on every modification done to the table, you must be careful to select columns for the disabled RELY constraint that will uniquely identify each row in the table. If such uniqueness is not present, then SQL Apply will not correctly maintain the table.

To improve the performance of SQL Apply, add a unique-constraint/index to the columns to identify the row on the logical standby database. Failure to do so results in full table scans during UPDATE or DELETE statements carried out on the table by SQL Apply.

See Also:

- See *Oracle Database Reference* for information about the DBA_LOGSTDBY_NOT_UNIQUE view
- *Oracle Database SQL Reference* for information about the ALTER TABLE statement syntax and creating RELY constraints
- [Section 9.6.1, "Create a Primary Key RELY Constraint"](#) on page 9-21 for information about RELY constraints and actions you can take to increase performance on a logical standby database

4.2 Step-by-Step Instructions for Creating a Logical Standby Database

This section describes the tasks you perform to create a logical standby database.

[Table 4–2](#) provides a checklist of the tasks that you perform to create a logical standby database and specifies on which database you perform each task. There is also a reference to the section that describes the task in more detail.

Table 4–2 *Creating a Logical Standby Database*

Reference	Task	Database
Section 4.2.1	Create a Physical Standby Database	Primary
Section 4.2.2	Stop Redo Apply on the Physical Standby Database	Standby
Section 4.2.3	Prepare the Primary Database to Support a Logical Standby Database	Primary
Section 4.2.4	Transition to a Logical Standby Database	Standby
Section 4.2.5	Open the Logical Standby Database	Standby
Section 4.2.6	Verify the Logical Standby Database Is Performing Properly	Standby

4.2.1 Create a Physical Standby Database

You create a logical standby database by first creating a physical standby database and then transitioning it to a logical standby database. Follow the instructions in [Chapter 3, "Creating a Physical Standby Database"](#) to create a physical standby database.

4.2.2 Stop Redo Apply on the Physical Standby Database

You can run Redo Apply on the new physical standby database for any length of time before converting it to a logical standby database. However, before converting to a logical standby database, stop Redo Apply on the physical standby database. Stopping Redo Apply is necessary to avoid applying changes past the redo that contains the LogMiner dictionary (described in [Section 4.2.3.2, "Build a Dictionary in the Redo Data"](#) on page 4-4).

To stop Redo Apply, issue the following statement on the physical standby database. If the database is a RAC database comprised of multiple instances, then you must first stop all RAC instances except one before issuing this statement:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE CANCEL;
```

4.2.3 Prepare the Primary Database to Support a Logical Standby Database

This section contains the following topics:

- [Prepare the Primary Database for Role Transitions](#)

- [Build a Dictionary in the Redo Data](#)

4.2.3.1 Prepare the Primary Database for Role Transitions

In [Section 3.1.4, "Set Primary Database Initialization Parameters"](#) on page 3-4, you set up several standby role initialization parameters to take effect when the primary database is transitioned to the *physical* standby role. If you plan to transition the primary database to the *logical* standby role, then you must also include a LOG_ARCHIVE_DEST_3 destination on the primary database, as shown in [Example 4-1](#), so that no parameters need to change after a role transition. This parameter only takes effect when the primary database is transitioned to the standby role.

Example 4-1 Primary Database: Logical Standby Role Initialization Parameters

```
LOG_ARCHIVE_DEST_3=
  'LOCATION=/arch2/chicago/
  VALID_FOR=(STANDBY_LOGFILES,STANDBY_ROLE)
  DB_UNIQUE_NAME=chicago'
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

To dynamically set the LOG_ARCHIVE_DEST_3 parameter, use the SQL ALTER SYSTEM SET statement and include the SCOPE=BOTH clause so that the change takes effect immediately and persists after the database is shut down and started up again.

The following table describes the archival processing defined by the initialization parameters shown in [Example 4-1](#).

	When the Chicago Database Is Running in the Primary Role	When the Chicago Database Is Running in the Logical Standby Role
LOG_ARCHIVE_DEST_3	Is ignored; LOG_ARCHIVE_DEST_3 is valid only when <code>chicago</code> is running in the standby role.	Archives redo data received from the primary database to the local archived redo log files in <code>/arch2/chicago/</code> .

4.2.3.2 Build a Dictionary in the Redo Data

A LogMiner dictionary must be built into the redo data so that the LogMiner component of SQL Apply can properly interpret changes it sees in the redo. As part of building LogMiner Multiversioned Data Dictionary, supplemental logging is automatically set up to log primary key and unique-constraint/index columns. The supplemental logging information ensures each update contains enough information to logically identify each row that is modified by the statement.

To build the LogMiner dictionary, issue the following statement:

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

The DBMS_LOGSTDBY.BUILD procedure waits for all existing transactions to complete. Long-running transactions executed on the primary database will affect the timeliness of this command.

The DBMS_LOGSTDBY.BUILD procedure uses Flashback Query to obtain a consistent snapshot of the data dictionary that is then logged in the redo stream. Oracle recommends setting the UNDO_RETENTION initialization parameter to 3600 on both the primary and logical standby databases.

See Also: The DBMS_LOGSTDBY.BUILD PL/SQL package in *Oracle Database PL/SQL Packages and Types Reference* and the UNDO_RETENTION initialization parameter in *Oracle Database Reference*

4.2.4 Transition to a Logical Standby Database

This section describes how to prepare the physical standby database to transition to a logical standby database. It contains the following topics:

- [Convert to a Logical Standby Database](#)
- [Create a New Password File](#)
- [Adjust Initialization Parameters for the Logical Standby Database](#)

4.2.4.1 Convert to a Logical Standby Database

The redo logs contain the information necessary to convert your physical standby database to a logical standby database. To continue applying redo data to the physical standby database until it is ready to convert to a logical standby database, issue the following SQL statement:

```
SQL> ALTER DATABASE RECOVER TO LOGICAL STANDBY db_name;
```

For *db_name*, specify a database name to identify the new logical standby database. If you are using a server parameter file (spfile) at the time you issue this statement, then the database will update the file with appropriate information about the new logical standby database. If you are not using an spfile, then the database issues a message reminding you to set the name of the DB_NAME parameter after shutting down the database.

The statement waits, applying redo data until the LogMiner dictionary is found in the log files. This may take several minutes, depending on how long it takes redo generated in [Section 4.2.3.2, "Build a Dictionary in the Redo Data"](#) to be transmitted to the standby database, and how much redo data need to be applied. If a dictionary build is not successfully performed on the primary database, this command will never complete. You can cancel the SQL statement by issuing the ALTER DATABASE RECOVER MANAGED STANDBY DATABASE CANCEL statement from another SQL session.

4.2.4.2 Create a New Password File

Because the conversion process changes the database name (that was originally set with the DB_NAME initialization parameter) for the logical standby database, you must re-create the password file. See *Oracle Database Administrator's Guide* for more information on creating a secure authentication scheme.

4.2.4.3 Adjust Initialization Parameters for the Logical Standby Database

On the logical standby database, shutdown the instance and issue the STARTUP MOUNT statement to start and mount the database. Do not open the database; it should remain closed to user access until later in the creation process. For example:

```
SQL> SHUTDOWN;
SQL> STARTUP MOUNT;
```

You need to modify the LOG_ARCHIVE_DEST_1 parameters because, unlike physical standby databases, logical standby databases are open databases that generate redo data and have multiple log files (online redo log files, archived redo log files, and standby redo log files). It is good practice to specify separate local destinations for:

- Archived redo log files that store redo data generated by the logical standby database. In [Example 4-2](#), this is configured as the LOG_ARCHIVE_DEST_1=LOCATION=/arch1/boston destination.

- Archived redo log files that store redo data received from the primary database. In [Example 4-2](#), this is configured as the LOG_ARCHIVE_DEST_3=LOCATION=/arch2/boston destination.

[Example 4-2](#) shows the initialization parameter changes that were modified for the logical standby database. The parameters shown are valid for the Boston logical standby database when it is running in either the primary or standby database role.

Example 4-2 Modifying Initialization Parameters for a Logical Standby Database

```
LOG_ARCHIVE_DEST_1=
'LOCATION=/arch1/boston/
VALID_FOR=(ONLINE_LOGFILES,ALL_ROLES)
DB_UNIQUE_NAME=boston'
LOG_ARCHIVE_DEST_2=
'SERVICE=chicago LGWR ASYNC
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)
DB_UNIQUE_NAME=chicago'
LOG_ARCHIVE_DEST_3=
'LOCATION=/arch2/boston/
VALID_FOR=(STANDBY_LOGFILES,STANDBY_ROLE)
DB_UNIQUE_NAME=boston'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_STATE_2=ENABLE
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

The following table describes the archival processing defined by the initialization parameters shown in [Example 4-2](#).

	When the Boston Database Is Running in the Primary Role	When the Boston Database Is Running in the Logical Standby Role
LOG_ARCHIVE_DEST_1	Directs archival of redo data generated by the primary database from the local online redo log files to the local archived redo log files in /arch1/boston/.	Directs archival of redo data generated by the logical standby database from the local online redo log files to the local archived redo log files in /arch1/boston/.
LOG_ARCHIVE_DEST_2	Directs transmission of redo data to the remote logical standby database chicago.	Is ignored; LOG_ARCHIVE_DEST_2 is valid only when boston is running in the primary role.
LOG_ARCHIVE_DEST_3	Is ignored; LOG_ARCHIVE_DEST_3 is valid only when boston is running in the standby role.	Directs archival of redo data received from the primary database to the local archived redo log files in /arch2/boston/.

Note: The DB_FILE_NAME_CONVERT initialization parameter is not honored once a physical standby database is converted to a logical standby database. If necessary, you should register a skip handler and provide SQL Apply with a replacement DDL string to execute by converting the path names of the primary database datafiles to the standby datafile path names. See the DBMS_LOGSTDBY package in *Oracle Database PL/SQL Packages and Types Reference*. for information about the SKIP procedure.

4.2.5 Open the Logical Standby Database

The new database is logically the same as your primary database, but it is transactionally inconsistent with the primary database, and thus incompatible for recovery operations.

To open the new logical standby database, you must open it with the `RESETLOGS` option by issuing the following statement:

```
SQL> ALTER DATABASE OPEN RESETLOGS;
```

Because this is the first time the database is being opened, the database's global name is adjusted automatically to match the new `DB_NAME` initialization parameter.

Issue the following statement to begin applying redo data to the logical standby database. For example:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE;
```

4.2.6 Verify the Logical Standby Database Is Performing Properly

See the following sections for help verifying that the logical standby database is performing properly:

- [Section 5.9.1, "Monitoring Log File Archival Information"](#) on page 5-29
- [Section 6.4.3, "Monitoring SQL Apply on Logical Standby Databases"](#) on page 6-6
- [Chapter 9, "Managing a Logical Standby Database"](#) on page 9-1

4.3 Post-Creation Steps

At this point, the logical standby database is running and can provide the maximum performance level of data protection. The following list describes additional preparations you can take on the logical standby database:

- Upgrade the data protection mode

The Data Guard configuration is initially set up in the maximum performance mode (the default). See [Section 5.6, "Setting Up a Data Protection Mode"](#) on page 5-18 for information about the data protection modes and how to upgrade or downgrade the current protection mode.

- Enable Flashback Database

Flashback Database removes the need to re-create the primary database after a failover. Flashback Database enables you to return a database to its state at a time in the recent past much faster than traditional point-in-time recovery, because it does not require restoring datafiles from backup nor the extensive application of redo data. You can enable Flashback Database on the primary database, the standby database, or both. See [Section 12.4, "Using Flashback Database After a Failover"](#) on page 12-23 and [Section 12.5, "Using Flashback Database After Issuing an Open Resetlogs Statement"](#) on page 12-27 for scenarios showing how to use Flashback Database in a Data Guard environment. Also, see *Oracle Database Backup and Recovery Advanced User's Guide* for more information about Flashback Database.

Redo Transport Services

This chapter describes how to configure redo transport services to transmit redo from a production database to one or more destinations. It contains the following topics:

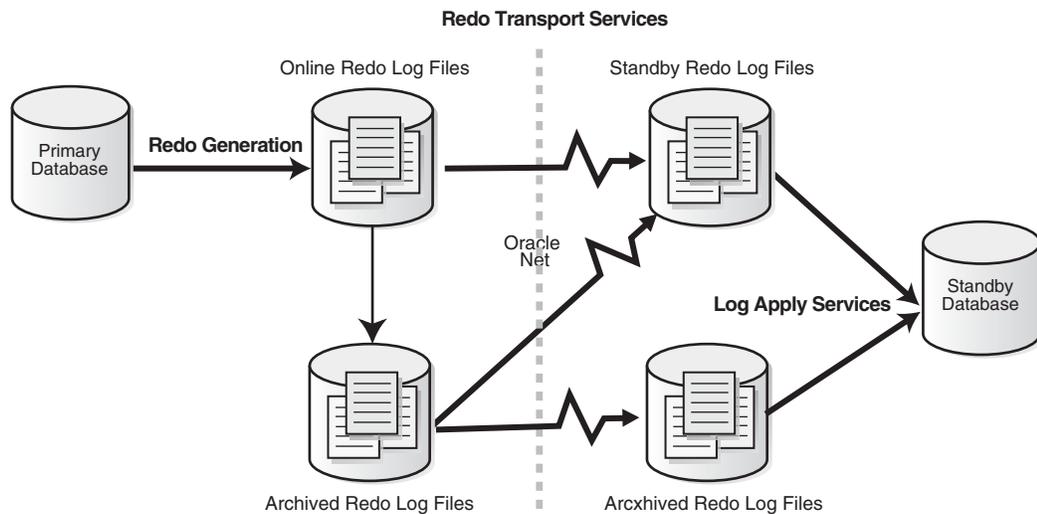
- [Introduction to Redo Transport Services](#)
- [Where to Send Redo Data](#)
- [How to Send Redo Data](#)
- [When Redo Data Should Be Sent](#)
- [What to Do If Errors Occur](#)
- [Setting Up a Data Protection Mode](#)
- [Managing Log Files](#)
- [Managing Archive Gaps](#)
- [Verification](#)

5.1 Introduction to Redo Transport Services

Redo transport services control the automated transfer of redo data from a database destination to one or more destinations. Redo transport services also manage the process of resolving any gaps in the archived redo log files due to a network failure.

Redo transport services can transmit redo data to local and remote destinations. Remote destinations can include any of the following types: physical and logical standby databases, archived redo log repositories, Oracle Change Data Capture staging databases, and Oracle Streams downstream capture databases.

[Figure 5–1](#) shows a simple Data Guard configuration with redo transport services archiving redo data to a local destination on the primary database while also transmitting it to archived redo log files or standby redo log files on a remote standby database destination.

Figure 5–1 Transmitting Redo Data

5.2 Where to Send Redo Data

This section contains the following topics:

- [Destination Types](#)
- [How to Send Redo Data](#)
- [Setting Up Flash Recovery Areas](#)

5.2.1 Destination Types

There are several types of destinations supported by redo transport services:

- Oracle Data Guard standby databases

Standby database destinations can be either physical standby databases or logical standby databases. [Section 1.1.2](#) discusses standby databases.

- Archived redo log repository

This type of destination allows off-site archiving of redo data. An archive log repository is created by using a physical standby control file, starting the instance, and mounting the database. This database contains no datafiles and cannot be used for switchover or failover. This alternative is useful as a way of holding archived redo log files for a short period of time, perhaps a day, after which the log files can then be deleted. This avoids most of the storage and processing expense of another fully configured standby database.

Oracle recommends using an archived redo log repository for temporary storage of archived redo files. This can be accomplished by configuring the repository destination for archiver-based transport (using the `ARCH` attribute on `LOG_ARCHIVE_DEST_n` parameter) in a Data Guard configuration running in maximum performance mode. For a no data loss environment, you should use a fully configured standby database using the `LGWR`, `SYNC`, and `AFFIRM` transport settings in a Data Guard configuration and running in either maximum protection mode or maximum availability mode.

- Oracle Streams real-time downstream capture database

This destination type allows Oracle Streams to configure a capture process on a remote downstream database. The Streams downstream capture process uses redo transport services to transfer redo data to the downstream database where a Streams capture process captures changes in the standby redo log files and archived redo log files on the remote destination. See *Oracle Streams Concepts and Administration* for more information.

- Oracle Change Data Capture staging database

This destination type supports a Change Data Capture Asynchronous AutoLog configuration remotely at a staging database. Redo data is copied from the source database to the staging database using redo transport services. The Change Data Capture configuration captures changes from the redo data. See *Oracle Database Data Warehousing Guide* for more information.

For discussion purposes, this guide refers to the production database as a primary database and to archival destinations as standby databases (as defined in [Section 1.1](#)). If you are using Oracle Change Data Capture, substitute the terms source and staging database for primary and standby database, respectively. If you are using Oracle Streams, substitute the terms source and downstream capture database for primary and standby database, respectively.

5.2.2 Configuring Destinations with the LOG_ARCHIVE_DEST_n Parameter

The LOG_ARCHIVE_DEST_n initialization parameter defines up to ten (where $n = 1, 2, 3, \dots, 10$) destinations, each of which must specify either the LOCATION or the SERVICE attribute to specify where to archive the redo data.

The LOCATION and SERVICE attributes describe either a local disk location or an Oracle Net service name that represents a standby destination to which redo transport services will transmit redo data. Specifying remote destinations with the SERVICE attribute allows Data Guard to maintain a transactionally consistent remote copy of the primary database for disaster recovery.

For every LOG_ARCHIVE_DEST_n initialization parameter that you define, specify a corresponding LOG_ARCHIVE_DEST_STATE_n parameter. The LOG_ARCHIVE_DEST_STATE_n (where n is an integer from 1 to 10) initialization parameter specifies whether the corresponding destination is currently on (enabled) or off (disabled). [Table 5-1](#) describes the LOG_ARCHIVE_DEST_STATE_n parameter attributes.

Table 5-1 LOG_ARCHIVE_DEST_STATE_n Initialization Parameter Attributes

Attribute	Description
ENABLE	Redo transport services can transmit redo data to this destination. This is the default.
DEFER	Redo transport services will not transmit redo data to this destination. This is a valid but unused destination.
ALTERNATE	This destination is not enabled, but it will become enabled if communication to its associated destination fails.
RESET	Functions the same as DEFER, but clears any error messages for the destination if it had previously failed.

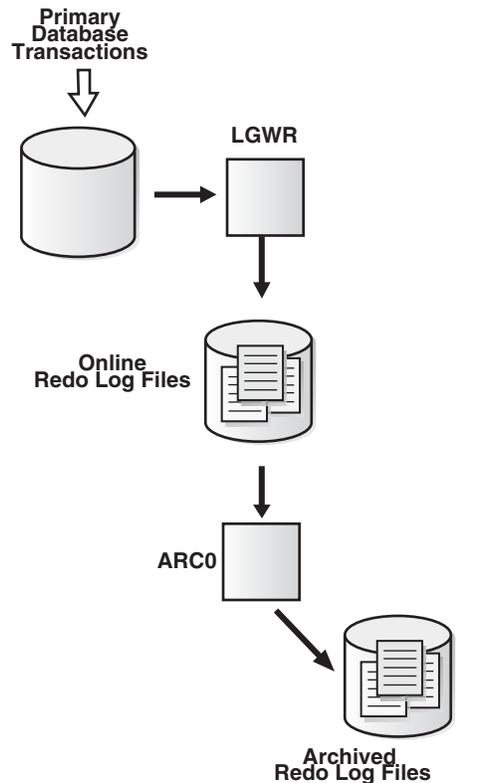
[Example 5-1](#) provides an example of one destination with the LOCATION attribute.

Example 5-1 Specifying a Local Archiving Destination

```
LOG_ARCHIVE_DEST_1='LOCATION=/arch1/chicago/'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
```

Figure 5–2 shows what this simple configuration, consisting of a single local destination, would look like. The log writer process writes redo data to the online redo log file. As each online redo log file is filled, a log switch occurs and an ARC_n process archives the filled online redo log file to an archived redo log file. The filled online redo log file is now available for reuse.

Figure 5–2 Primary Database Archiving When There Is No Standby Database



It is important to note that the configuration shown in Figure 5–2 does not include a standby database and thus does not provide disaster-recovery protection. To make this simple configuration into a Data Guard configuration that provides disaster recovery, add a standby database at a remote destination by specifying the `SERVICE` attribute.

Example 5–2 shows the initialization parameters that enable redo transport services to archive the online redo log on the local destination `chicago` and transmit redo data to a remote standby database with the Oracle Net service name `boston`. The example takes the default values for all of the other `LOG_ARCHIVE_DEST_n` attributes:

Example 5–2 Specifying a Remote Archiving Destination

```
LOG_ARCHIVE_DEST_1='LOCATION=/arch1/chicago/'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_2='SERVICE=boston'
LOG_ARCHIVE_DEST_STATE_2=ENABLE
```

These initialization parameters set up a Data Guard configuration that uses archiver (ARC_n) processes to archive to both the local and remote destinations. This configuration provides the maximum performance level of data protection.

Although you can create a Data Guard configuration by specifying only the `LOCATION` or the `SERVICE` attributes on the `LOG_ARCHIVE_DEST_n` parameter, you can optionally specify more attributes to further define each destination's behavior. [Chapter 14](#) provides reference information for all of the `LOG_ARCHIVE_DEST_n` parameter attributes.

5.2.2.1 Changing Destination Attributes

You can dynamically update most of the attribute values of the `LOG_ARCHIVE_DEST_n` and the `LOG_ARCHIVE_DEST_STATE_n` parameters using the `ALTER SYSTEM SET` statement.

The modifications take effect after the next log switch on the primary database. For example, to defer redo transport services from transmitting redo data to the remote standby database named `boston`, issue the following statements on the primary database:

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_2='SERVICE=boston
  2> VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)';
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2=DEFER;
```

5.2.2.2 Viewing Attribute with V\$ARCHIVE_DEST

Query the `V$ARCHIVE_DEST` view to see current settings of the `LOG_ARCHIVE_DEST_n` initialization parameter.

5.2.3 Setting Up Flash Recovery Areas

The Oracle database enables you to configure a disk area called a *flash recovery area* that is a directory or Oracle Storage Manager disk group that serves as the default storage area for files related to recovery.

To configure a flash recovery area, use the `DB_RECOVERY_FILE_DEST` initialization parameter. `LOG_ARCHIVE_DEST_10` is implicitly set to `USE_DB_RECOVERY_FILE_DEST` (meaning that archived redo log files will be sent to the flash recovery area) if you create a recovery area and do not set any other local archiving destinations. (See *Oracle Database Backup and Recovery Basics* to configure the flash recovery area and *Oracle Database Administrator's Guide* for more information about Oracle Storage Manager and Oracle Managed Files.)

Note: The filenames for archived redo log files stored in a flash recovery area are generated automatically by Oracle Managed Files (OMF); the filenames are not based on the format specified by the `LOG_ARCHIVE_FORMAT` initialization parameter.

This section contains the following topics:

- [Using the LOG_ARCHIVE_DEST_10 Destination](#)
- [Using Other LOG_ARCHIVE_DEST_n Destinations](#)
- [Using the STANDBY_ARCHIVE_DEST Destination](#)
- [Sharing a Flash Recovery Area Between Primary and Standby Databases](#)

Note: A primary database cannot write redo data to the flash recovery area of a logical standby database.

See *Oracle Database Backup and Recovery Basics* to configure flash recovery areas and [Section 10.3.4](#) for information about setting up a deletion policy for archived redo log files in flash recovery areas.

5.2.3.1 Using the LOG_ARCHIVE_DEST_10 Destination

If a flash recovery area has been configured and no local destinations are defined, Data Guard implicitly uses the LOG_ARCHIVE_DEST_10 destination as the flash recovery area.

When the LOG_ARCHIVE_DEST_10 destination is used, Data Guard automatically uses the default values for all of the LOG_ARCHIVE_DEST_10 parameter attributes. To override the defaults, you can dynamically set the values for most of the attributes by explicitly specifying the LOG_ARCHIVE_DEST_10 parameter. For example, the following ALTER SYSTEM SET statement specifies several attributes on the LOG_ARCHIVE_DEST_10 initialization parameter:

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_10='LOCATION=USE_DB_RECOVERY_FILE_DEST LGWR
MANDATORY REOPEN=5 VALID_FOR=(ONLINE_LOGFILES, PRIMARY_ROLE) '
```

When setting LOG_ARCHIVE_DEST_10 attributes, the TEMPLATE attribute of a LOG_ARCHIVE_DEST_10 parameter will override all other specifications for the flash recovery area. If the TEMPLATE attribute is specified for a remote destination and that destination archives redo data to a flash recovery area, the archived redo log file will use the directory and file name specified by the TEMPLATE attribute.

5.2.3.2 Using Other LOG_ARCHIVE_DEST_n Destinations

You can explicitly set up one or more other LOG_ARCHIVE_DEST_n destinations to point to a flash recovery area. For example, you can optionally:

- Configure destinations other than LOG_ARCHIVE_DEST_10

For example, an existing Data Guard configuration may have already used the LOG_ARCHIVE_DEST_10 destination for another purpose, or you may want to release the LOG_ARCHIVE_DEST_10 destination for other uses.

To configure another archival destination to point to the flash recovery area, you must specify the LOCATION=USE_DB_RECOVERY_FILE_DEST attribute to define the new destination. For example:

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_1='LOCATION=USE_DB_RECOVERY_FILE_DEST
ARCH MANDATORY REOPEN=5 VALID_FOR=(ONLINE_LOGFILES, PRIMARY_ROLE) '
```

The implicit setting (for LOG_ARCHIVE_DEST_10 to use the flash recovery area) will be cleared.

- Configure destinations in addition to LOG_ARCHIVE_DEST_10 destination for use after a role transition

For example, you can configure one destination to be valid for standby redo log archival when the database operates in the standby role and another destination to be valid for online redo log archival when the database operates in the primary role.

To configure a LOG_ARCHIVE_DEST_n destination in addition to LOG_ARCHIVE_DEST_10, you must explicitly specify both destinations:

```
LOG_ARCHIVE_DEST_9='LOCATION=USE_DB_RECOVERY_FILE_DEST ARCH MANDATORY REOPEN=5
VALID_FOR=(STANDBY_LOGFILES, STANDBY_ROLE) '
LOG_ARCHIVE_DEST_10='LOCATION=USE_DB_RECOVERY_FILE_DEST ARCH MANDATORY REOPEN=5
VALID_FOR=(ONLINE_LOGFILES, PRIMARY_ROLE) '
```

5.2.3.3 Using the STANDBY_ARCHIVE_DEST Destination

On a physical standby database, you can define the `STANDBY_ARCHIVE_DEST` parameter to point to the flash recovery area. For example:

```
STANDBY_ARCHIVE_DEST='LOCATION=USE_DB_RECOVERY_FILE_DEST'
```

Note: Flash recovery area destinations pointed to by the `STANDBY_ARCHIVE_DEST` parameter on logical standby databases (SQL Apply) are ignored.

5.2.3.4 Sharing a Flash Recovery Area Between Primary and Standby Databases

You can share a flash recovery area between databases provided each database that shares the flash recovery area has a unique database name, specified with the `DB_UNIQUE_NAME` initialization parameter.

The following examples show how to specify initialization parameters on the primary and standby databases that will share a flash recovery area in the `/arch/oradata` location. Although the `DB_UNIQUE_NAME` parameter is not specified in [Example 5–3](#), it defaults to `PAYROLL`, which is the name specified for the `DB_NAME` initialization parameter.

Example 5–3 Primary Database Initialization Parameters for a Shared Recovery Area

```
DB_NAME=PAYROLL
LOG_ARCHIVE_DEST_1='LOCATION=USE_DB_RECOVERY_FILE_DEST'
DB_RECOVERY_FILE_DEST='/arch/oradata'
DB_RECOVERY_FILE_DEST_SIZE=20G
```

Example 5–4 Standby Database Initialization Parameters for a Shared Recovery Area

```
DB_NAME=PAYROLL
DB_UNIQUE_NAME=boston
LOG_ARCHIVE_DEST_1='LOCATION=USE_DB_RECOVERY_FILE_DEST'
STANDBY_ARCHIVE_DEST='LOCATION=USE_DB_RECOVERY_FILE_DEST'
DB_RECOVERY_FILE_DEST='/arch/oradata'
DB_RECOVERY_FILE_DEST_SIZE=5G
```

See *Oracle Database Backup and Recovery Advanced User's Guide* for more information about sharing a flash recovery area among multiple databases.

5.3 How to Send Redo Data

On the primary database, Oracle Data Guard uses archiver processes (`ARCn`) or the log writer process (`LGWR`) to collect transaction redo data and transmit it to standby destinations. Although you cannot use both the archiver and log writer processes to send redo data to the same destination, you can choose to use the log writer process for some destinations, while archiver processes send redo data to other destinations.

This section contains the following topics:

- [Using Archiver Processes \(ARCn\) to Archive Redo Data](#)
- [Using the Log Writer Process \(LGWR\) to Archive Redo Data](#)
- [Providing for Secure Redo Data Transmission](#)

Data Guard also uses the fetch archive log (FAL) client and server to send archived redo log files to standby destinations following a network outage, for automatic gap resolution, and resynchronization. The FAL process and gap resolution are discussed in [Section 5.8](#).

5.3.1 Using Archiver Processes (ARCn) to Archive Redo Data

By default, redo transport services use ARC*n* processes to archive the online redo log files on the primary database. ARC*n* archival processing supports only the maximum performance level of data protection in Data Guard configurations. You must use the LGWR process to transmit redo data to standby locations that operate in other data protection modes. (See [Section 5.6](#) for more information about the Data Guard data protection modes.)

The following sections discuss these topics:

- [Initialization Parameters That Control ARC*n* Archival Behavior](#)
- [ARC*n* Archival Processing](#)

5.3.1.1 Initialization Parameters That Control ARC*n* Archival Behavior

The following descriptions tell how to use the LOG_ARCHIVE_DEST_*n* and the LOG_ARCHIVE_MAX_PROCESSES initialization parameters.

Enabling ARC*n* Processes to Archive to Local or Remote Destinations

You specify attributes on the LOG_ARCHIVE_DEST_*n* initialization parameter to control the automated transfer of redo data from the primary database to other destinations. Because ARC*n* archiver processing is the default archival behavior, specifying the ARCH attribute on the LOG_ARCHIVE_DEST_*n* parameter is optional. However, you must specify either the LOCATION attribute to archive to a local destination or the SERVICE attribute for remote archival (as described in [Section 5.2.2](#)).

Specifying the Number of ARC*n* Processes to be Invoked

The LOG_ARCHIVE_MAX_PROCESSES initialization parameter specifies the maximum number of ARC*n* processes. By default, 4 archiver processes are invoked when the primary database instance starts and Oracle Database dynamically adjusts the number of processes to balance the archiver workload. Thus, the actual number of archiver processes may vary at any time.

If you anticipate a heavy workload for archiving, you can increase the maximum number of archiver processes to as many as 30 by setting the initialization parameter LOG_ARCHIVE_MAX_PROCESSES. This initialization parameter is dynamic and can be altered by the ALTER SYSTEM command to increase or decrease the maximum number of archiver processes. For example:

```
ALTER SYSTEM SET LOG_ARCHIVE_MAX_PROCESSES = 20;
```

5.3.1.2 ARC*n* Archival Processing

[Figure 5–3](#) shows an example of archival processing in a Data Guard configuration, with a primary database located in Chicago and one physical standby database located in Boston.

Archiving happens when a log switch occurs on the primary database:

- On the primary database, after the ARC0 process successfully archives the local online redo log to the local destination (LOG_ARCHIVE_DEST_1), the ARC1

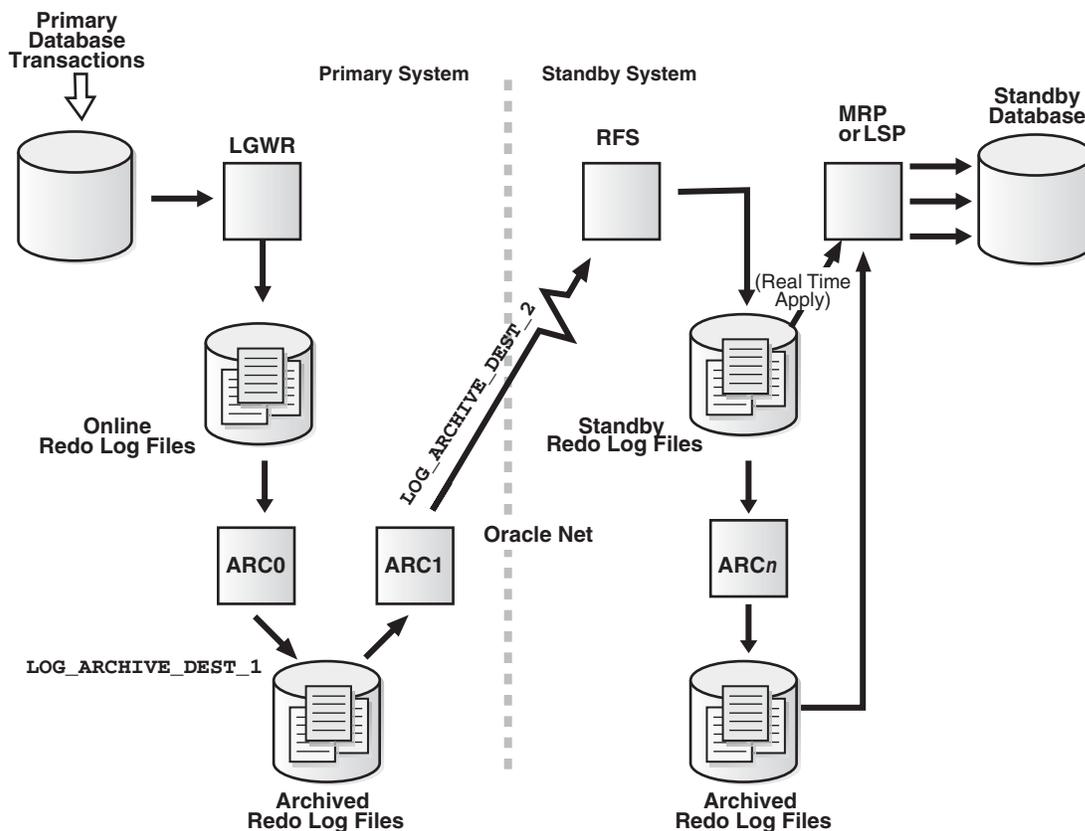
process transmits redo from the local archived redo log files (instead of the online redo log files) to the remote standby destination (LOG_ARCHIVE_DEST_2).

- On the remote destination, the remote file server process (RFS) will, in turn, write the redo data to an archived redo log file from a standby redo log file. Log apply services use Redo Apply (MRP process¹) or SQL Apply (LSP process²) to apply the redo to the standby database.

Because the online redo log files are archived locally first, the LGWR process reuses the online redo log files much earlier than would be possible if the ARC_n processes archived to the standby database concurrently with the local destination.

As shown in Figure 5-3, you need to have at least 2 ARC_n processes to separate local archival from remote archival. This can be done by setting the LOG_ARCHIVE_MAX_PROCESSES initialization parameter (the default setting is 4, but the maximum value is 30).

Figure 5-3 Archiving to Local Destinations Before Archiving to Remote Destinations



Because the default ARC_n archival processing disassociates local archiving from remote archiving, sites that may have policies to delete archived redo log files on the primary database immediately after backing them up must make sure that the standby destinations receive the corresponding redo data before deleting the archived redo log

¹ The managed recovery process (MRP) applies archived redo log files to the physical standby database, and automatically determines the optimal number of parallel recovery processes at the time it starts. The number of parallel recovery slaves spawned is based on the number of CPUs available on the standby server.

² The logical standby process (LSP) uses parallel execution (P_{nnn}) processes to apply archived redo log files to the logical standby database, using SQL interfaces.

files on the primary database. You can query the `V$ARCHIVED_LOG` view to verify the redo data was received on standby destinations.

5.3.2 Using the Log Writer Process (LGWR) to Archive Redo Data

You can optionally enable redo transport services to use the LGWR process to transmit redo data to remote destinations.

Using the LGWR process differs from `ARCn` processing (described in [Section 5.3.1](#)), because instead of waiting for the online redo log to switch at the primary database and then writing the entire archived redo log at the remote destination all at once, the LGWR process selects a standby redo log file at the standby site that reflects the log sequence number (and size) of the current online redo log file of the primary database. Then, as redo is generated at the primary database, it is also transmitted to the remote destination. The transmission to the remote destination will either be synchronous or asynchronous, based on whether the `SYNC` or the `ASync` attribute is set on the `LOG_ARCHIVE_DEST_n` parameter. Synchronous LGWR processing is required for the maximum protection and maximum availability modes of data protection in Data Guard configurations. (See [Section 5.6](#) for information about the Data Guard data protection modes.)

This section contains the following topics:

- [LOG_ARCHIVE_DEST_n Attributes for LGWR Archival Processing](#)
- [LGWR SYNC Archival Processing](#)
- [LGWR ASync Archival Processing](#)

5.3.2.1 LOG_ARCHIVE_DEST_n Attributes for LGWR Archival Processing

The following sections describe the `LGWR`, `SYNC`, and `ASync` attributes.

Enabling Redo Transport Services to Use the LGWR Process

You must specify the `LGWR` and `SERVICE` attributes on the `LOG_ARCHIVE_DEST_n` parameter to enable redo transport services to use the LGWR process to transmit redo data to remote archival destinations.

Specifying Synchronous or Asynchronous Network Transmission

The LGWR process synchronously writes to the local online redo log files at the same time it transmits redo data to the remote destination:

- The `SYNC` attribute performs all network I/O synchronously, in conjunction with each write operation to the online redo log file, and waits for the network I/O to complete. [Section 5.3.2.2](#) shows an example of synchronous network transmission in a Data Guard configuration. This is the default network transmission setting.
- The `ASync` attribute performs all network I/O asynchronously and control is returned to the executing application or user immediately, without waiting for the network I/O to complete. [Section 5.3.2.3](#) shows an example of asynchronous network transmission in a Data Guard configuration.

Note: If you configure a destination to use the LGWR process, but for some reason the LGWR process becomes unable to archive to the destination, then redo transport will revert to using the `ARCn` process to complete archival operations.

5.3.2.2 LGWR SYNC Archival Processing

[Example 5-5](#) shows the primary role LOG_ARCHIVE_DEST_ *n* parameters that configure the LGWR process for synchronous network transmission.

Example 5-5 Initialization Parameters for LGWR Synchronous Archival

```
LOG_ARCHIVE_DEST_1='LOCATION=/arch1/chicago'
LOG_ARCHIVE_DEST_2='SERVICE=boston LGWR SYNC NET_TIMEOUT=30'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_STATE_2=ENABLE
```

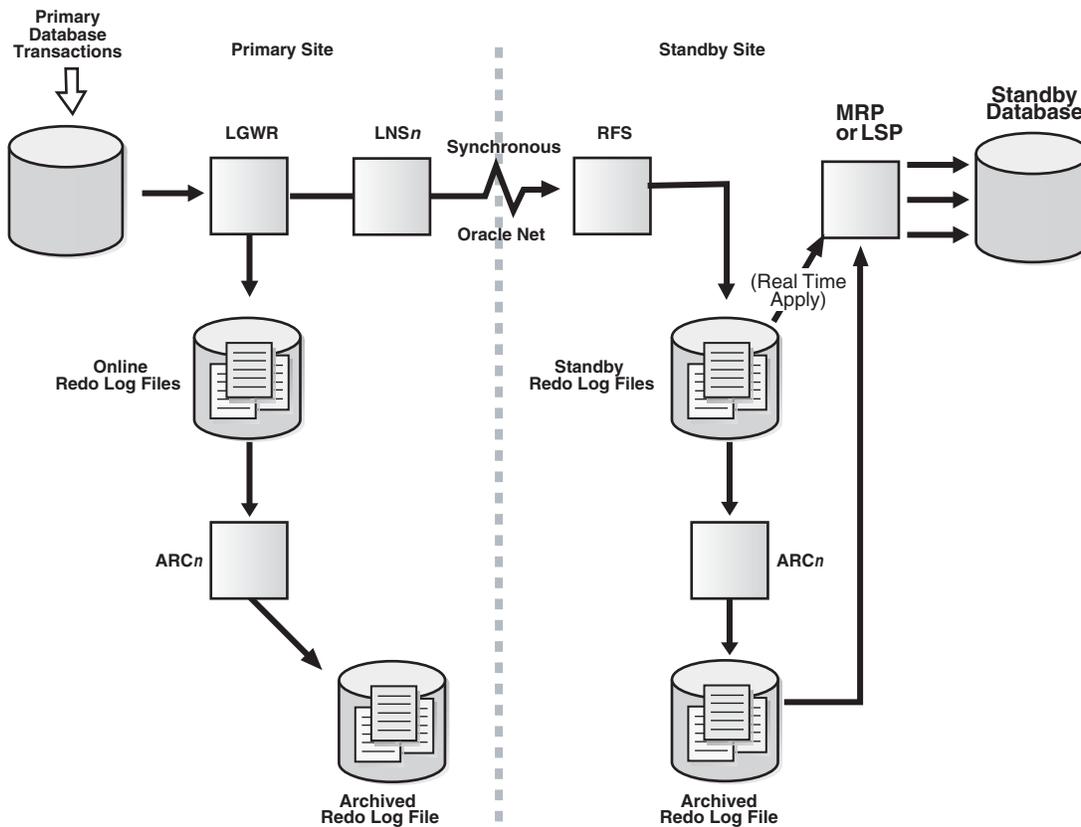
Specifying the SYNC attribute on the LOG_ARCHIVE_DEST_ *n* parameter is optional, because this is the default for LGWR archival processing. The NET_TIMEOUT attribute is recommended, because it controls the amount of time that the LGWR process waits for status from the network server process before terminating the network connection. If there is no reply within NET_TIMEOUT seconds, then the LGWR process returns an error message.

[Figure 5-4](#) shows a Data Guard configuration that uses the LGWR process to synchronously transmit redo data to the standby system at the same time it is writing redo data to the online redo log file on the primary database:

- On the primary database, the LGWR process submits the redo data to one or more network server (LNS_{*n*}) processes, which then initiate the network I/O in parallel to multiple remote destinations. Transactions are not committed on the primary database until the redo data necessary to recover the transaction is received by all LGWR SYNC destinations.
- On the standby system, the remote file server (RFS) receives redo data over the network from the LGWR process and writes the redo data to the standby redo log files.

A log switch on the primary database triggers a log switch on the standby database, causing ARC_{*n*} processes on the standby database to archive the standby redo log files to archived redo log files on the standby database. Then, Redo Apply (MRP process) or SQL Apply (LSP process) applies the redo data to the standby database. If real-time apply is enabled, Data Guard recovers redo data directly from the current standby redo log file as it is being filled up by the RFS process.

Figure 5-4 LGWR SYNC Archival to a Remote Destination with Standby Redo Log Files



5.3.2.3 LGWR ASYNC Archival Processing

Example 5-6 shows the primary role LOG_ARCHIVE_DEST_n parameters that configure the LGWR process for asynchronous network transmission.

Example 5-6 Initialization Parameters for LGWR Asynchronous Archiving

```
LOG_ARCHIVE_DEST_1='LOCATION=/arch1/chicago'
LOG_ARCHIVE_DEST_2='SERVICE=boston LGWR ASYNC'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_STATE_2=ENABLE
```

Figure 5-5 shows the LNSn process collecting redo data from the online redo log files and transmitting it over Oracle Net to the RFS process on the standby database.

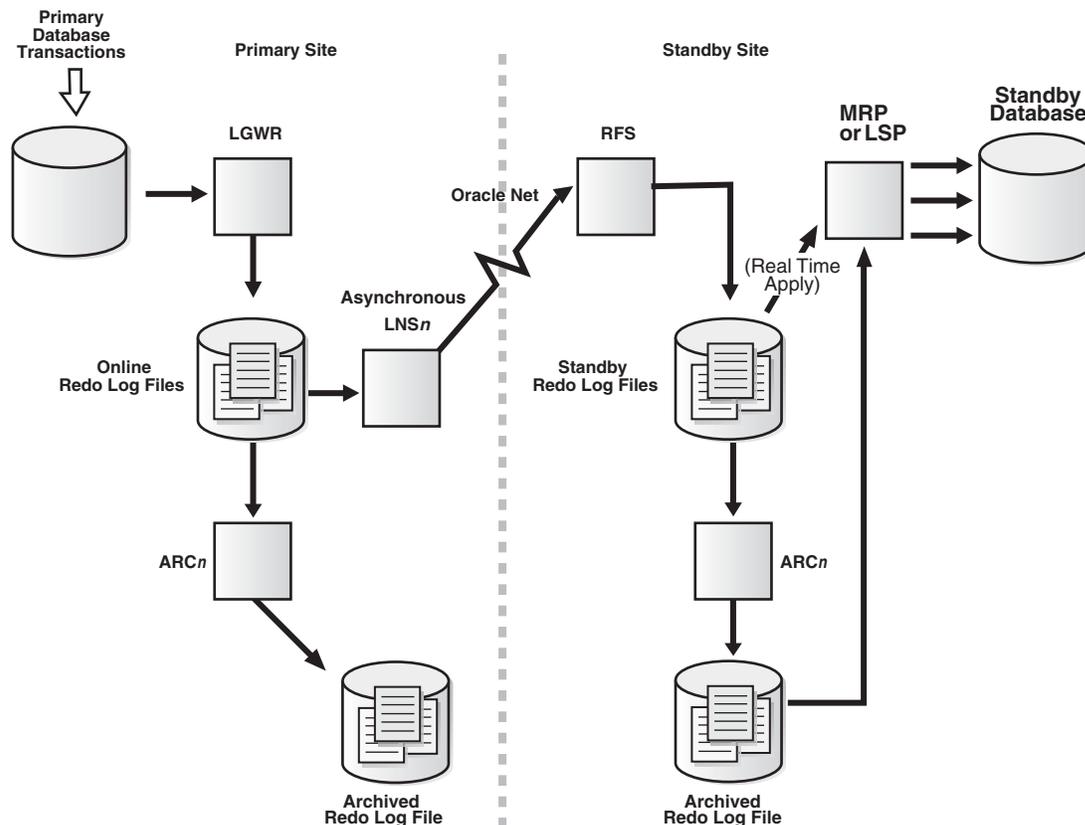
When the LGWR and ASYNC attributes are specified, the log writer process writes to the local online redo log file, while the network server (LNSn) processes (one for each destination) asynchronously transmit the redo to remote destinations. The LGWR process continues processing the next request without waiting for the LNS network I/O to complete.

If redo transport services transmit redo data to multiple remote destinations, the LNSn processes (one for each destination) initiate the network I/O to all of the destinations in parallel.

When an online redo log file fills up, a log switch occurs and an archiver process archives the log file locally, as usual.

Note: Beginning with Oracle Database 10g Release 10.2, it is unnecessary to specify the `NET_TIMEOUT` attribute on the `LOG_ARCHIVE_DEST_n` destinations configured with both the `LGWR` and `ASYNC` attributes. This is because the log writer process never waits for the `LNSn` for any reason in release 10.2. Thus, specifying the `NET_TIMEOUT` attribute is not required.

Figure 5–5 LGWR ASYNC Archival with Network Server (LNS_n) Processes



Note: Do not use the `LOG_ARCHIVE_DEST` or the `LOG_ARCHIVE_DUPLEX_DEST` initialization parameters to specify a flash recovery area destination.

5.3.3 Providing for Secure Redo Data Transmission

Data Guard provides a secure environment and prevents the possible tampering of redo data as it is being transferred to the standby database.

Redo transport services use authenticated network sessions to transfer redo data. These sessions are authenticated using the SYS user password contained in the password file. All databases in the Data Guard configuration must use a password file, and the SYS password contained in this password file must be identical on all systems. This authentication can be performed even if Oracle Advanced Security is not installed, and provides some level of security when shipping redo.

Note: To further protect redo (for example, to encrypt redo or compute an integrity checksum value for redo traffic over the network to disallow redo tampering on the network), Oracle recommends that you install and use Oracle Advanced Security. See the *Oracle Database Advanced Security Administrator's Guide*.

Perform the following steps on the primary database and each standby database:

1. Create a password file (using the `orapwd` utility) on the primary and all standby databases. For example:

```
ORAPWD FILE=orapw PASSWORD=mypassword ENTRIES=10
```

This example creates a password file with 10 entries, where the password for SYS is *mypassword*. For redo data transmission to succeed, ensure you set the password for the SYS user account identically for every primary and standby database.

2. Set the `REMOTE_LOGIN_PASSWORDFILE` initialization parameter to `EXCLUSIVE` or `SHARED` to enable Oracle to check for a password file and to specify how many databases can use the password file. For example:

```
REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE
```

See the *Oracle Database Reference* for more information about this parameter.

5.4 When Redo Data Should Be Sent

This section contains the following topics:

- [Specifying Role-Based Destinations with the VALID_FOR Attribute](#)
- [Specify Unique Names for Primary and Standby Databases](#)

5.4.1 Specifying Role-Based Destinations with the VALID_FOR Attribute

The `VALID_FOR` attribute enables you to configure destination attributes for both the primary and standby database roles in one server parameter file (SPFILE), so that your Data Guard configuration operates properly after a role transition. This simplifies switchovers and failovers by removing the need to enable and disable the role-specific parameter files after a role transition.

When you specify the `VALID_FOR` attribute of the `LOG_ARCHIVE_DEST_n` parameter, it identifies when redo transport services can transmit redo data to destinations based on the following factors:

- Whether the database is *currently* running in the primary or the standby role
- Whether archival of the online redo log file, standby redo log file, or both is required depending on the *current* role of the database

To configure these factors for each `LOG_ARCHIVE_DEST_n` destination, you specify this attribute with a pair of keywords: `VALID_FOR=(redo_log_type, database_role)`. The `redo_log_type` keyword identifies the destination as valid for archiving the following: `ONLINE_LOGFILE`, `STANDBY_LOGFILE`, or `ALL_LOGFILES`. The `database_role` keyword identifies the role in which the current database must be in for the destination to be valid: `PRIMARY_ROLE`, `STANDBY_ROLE`, or `ALL_ROLES`.

If you do not specify the `VALID_FOR` attribute for a destination, by default, archiving the online redo log and standby redo log is enabled to the destination, regardless of

the database role. This default behavior is equivalent to setting the (ALL_LOGFILES, ALL_ROLES) keyword pair on the VALID_FOR attribute. For example:

```
LOG_ARCHIVE_DEST_1='LOCATION=/ARCH1/CHICAGO/ VALID_FOR=(ALL_LOGFILES, ALL_ROLES)'
```

Although the (ALL_LOGFILES, ALL_ROLES) keyword pair is the default, it is not recommended for every destination. For example, logical standby databases, unlike physical standby databases, are open databases that generate redo data and have multiple log files (online redo log files, archived redo log files, and standby redo log files). In most cases, the online redo log files generated by the logical standby database are located in the same directory as the standby redo logs files that are receiving redo from the primary database.

Therefore, it is recommended that you define a VALID_FOR attribute for each destination so that your Data Guard configuration operates properly, including after a role transition. See the scenarios in [Section 12.1](#) for examples of the VALID_FOR attribute settings for various Data Guard configurations,

If you choose not to use the VALID_FOR attribute to configure destinations, you must maintain two database server parameter files (SPFILEs) for each database: one for when the database is in the primary role and the other for the standby role. See [Chapter 12](#) for more configuration examples.

5.4.2 Specify Unique Names for Primary and Standby Databases

The DB_UNIQUE_NAME attribute enables you to specify unique database names when you configure destinations. This makes it possible to dynamically add a standby database to a Data Guard configuration that contains a Real Applications Clusters primary database, when that primary database is operating in either the maximum protection or the maximum availability level of protection. The DB_UNIQUE_NAME initialization parameter is required if the LOG_ARCHIVE_CONFIG parameter has been defined.

Note: If the standby database on a remote destination has not been identified using the DB_UNIQUE_NAME initialization parameter, the standby database must be accessible before the primary instance is started.

Together, the DB_UNIQUE_NAME attribute of the LOG_ARCHIVE_DEST_n parameter and the DG_CONFIG attribute of the LOG_ARCHIVE_CONFIG parameter specify the unique name of each database of the Data Guard configuration. The names you supply must match what was defined for each database with the DB_UNIQUE_NAME initialization parameter.

For example, the following initialization parameters show the DB_UNIQUE_NAME and LOG_ARCHIVE_CONFIG definitions for the primary database (chicago) in the Data Guard configuration described in [Chapter 3](#):

```
DB_NAME=chicago
DB_UNIQUE_NAME=chicago
LOG_ARCHIVE_CONFIG='DG_CONFIG=(chicago, boston) '
LOG_ARCHIVE_DEST_1='LOCATION=/arch1/chicago/ VALID_FOR=(ALL_LOGFILES, ALL_ROLES)
LOG_ARCHIVE_DEST_2=
'SERVICE=boston LGWR ASYNC
VALID_FOR=(ONLINE_LOGFILES, PRIMARY_ROLE)
DB_UNIQUE_NAME=boston'
```

The `DB_UNIQUE_NAME` attribute is required for remote destinations specified with the `SERVICE` attribute. In the example, the `LOG_ARCHIVE_DEST_2` parameter specifies the `DB_UNIQUE_NAME=boston` for the remote destination; redo transport services validate this information at the remote destination. If the names do not match, the connection to that destination is refused.

The `LOG_ARCHIVE_CONFIG` parameter also has `SEND`, `NOSEND`, `RECEIVE`, and `NORECEIVE` attributes:

- `SEND` enables a database to send redo data to remote destinations
- `RECEIVE` enables the standby database to receive redo from another database

To disable these settings, use the `NOSEND` and `NORECEIVE` keywords.

For example, to ensure the primary database never accidentally receives any archived redo data, set the `LOG_ARCHIVE_CONFIG` initialization parameter to `NORECEIVE` on the primary database, as follows:

```
LOG_ARCHIVE_CONFIG='NORECEIVE,DG_CONFIG=(chicago,boston)'
```

However, keep in mind that specifying either the `NOSEND` or the `NORECEIVE` attributes may limit the database instance's capabilities after a role transition. For example, if a standby database with the `NOSEND` attribute set is transitioned to the primary role, it would not be able to transmit redo data to other standby databases until you reset the parameter value to `SEND`. Similarly, a database that has the `NORECEIVE` attribute specified cannot receive redo from the primary database.

By default, the `LOG_ARCHIVE_CONFIG` parameter allows the primary database to send redo data to the standby database and allows the standby database to receive redo from the primary database for archiving. This is equivalent to setting both `SEND` and `RECEIVE` attributes on the `LOG_ARCHIVE_CONFIG` parameter.

Note: The `LOG_ARCHIVE_CONFIG` initialization parameter replaces the `REMOTE_ARCHIVE_ENABLE` initialization parameter, which is deprecated. Do not specify both parameters in the same SPFILE or text initialization parameter file.

5.5 What to Do If Errors Occur

To handle archiving failures, you can use the `REOPEN`, `MAX_FAILURES`, and `ALTERNATE` attributes of the `LOG_ARCHIVE_DEST_n` parameter to specify what actions are to be taken when archival processing to a destination fails. These actions include:

- Retrying the archival operation to a failed destination after a specified period of time, up to a limited number of times
- Using an alternate or substitute destination
- Controlling the number of attempts to reestablish communication and resume sending redo data to a failed destination.

5.5.1 Retrying the Archival Operation

Use the `REOPEN` attribute to determine if and when the `ARCn` process or the `LGWR` process attempts to transmit redo data again to a failed destination following an error.

Use the `REOPEN=seconds` attribute to specify the minimum number of seconds that must elapse following an error before the archiving process will try again to access a

failed destination. The default value is 300 seconds. The value set for the `REOPEN` attribute applies to all errors, not just connection failures. You can turn off the option by specifying `REOPEN=0`, which prevents the destination from being retried after a failure occurs.

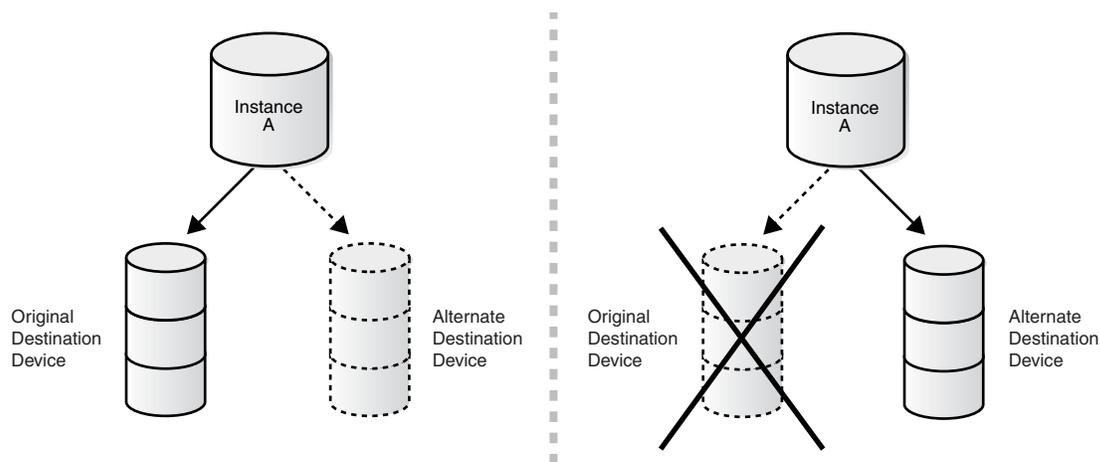
If transmission to the alternate destination fails and the `REOPEN` attribute is set to zero (0), redo transport services will attempt to send redo data to the alternate destination the next time redo data is archived.

5.5.2 Using an Alternate Destination

The `ALTERNATE` attribute defines an alternate archiving destination that can be used when the original archiving destination fails. If no alternate destination is specified, the destination does not automatically change to another destination upon failure.

Figure 5–6 shows a scenario where redo data is archived to a local disk device. If the original destination device becomes full or unavailable, the archival operation is automatically redirected to the alternate destination device.

Figure 5–6 Archival Operation to an Alternate Destination Device



The `REOPEN` attribute takes precedence over the `ALTERNATE` attribute. The alternate destination is used only if one of the following conditions is true:

- A value of zero (0) is specified for the `REOPEN` attribute.
- A nonzero `REOPEN` attribute and a nonzero `MAX_FAILURE` count have been exceeded.

The `ALTERNATE` attribute takes precedence over the `MANDATORY` attribute. This means that a destination fails over to a valid alternate destination even if the current destination is mandatory.

See Also: The [ALTERNATE](#) attribute on page 14-4

5.5.3 Controlling the Number of Retry Attempts

Use the `MAX_FAILURE` attribute to specify the maximum number of consecutive times that redo transport services attempt to transmit redo data to a failed destination. To limit the number of consecutive attempts that will be made to reestablish communication with a failed destination, use the `REOPEN` attribute in conjunction with

the `MAX_FAILURE` attribute. Once the specified number of consecutive attempts is exceeded, the destination is treated as if the `REOPEN` attribute was set to zero.

The `REOPEN` attribute is required when you use the `MAX_FAILURE` attribute. [Example 5-7](#) shows how to set a retry time of 60 seconds and limit retries to 3 attempts.

Example 5-7 Setting a Retry Time and Limit

```
LOG_ARCHIVE_DEST_1='LOCATION=/arc_dest REOPEN=60 MAX_FAILURE=3'
```

5.6 Setting Up a Data Protection Mode

Data Guard provides three modes of data protection: maximum protection, maximum availability, and maximum performance. The level of data protection you choose controls what happens if the primary database loses its connection to the standby database. This section contains the following topics:

- [Choosing a Data Protection Mode](#)
- [Setting the Data Protection Mode of a Data Guard Configuration](#)

5.6.1 Choosing a Data Protection Mode

To determine the appropriate data protection mode to use, review the following descriptions of the data protection modes to help assess your business requirements for data availability against user demands for response time and performance. Also, see [Section 5.6.2](#) for information about setting up the data protection mode.

5.6.1.1 Maximum Protection Mode

This protection mode ensures that no data loss will occur if the primary database fails. To provide this level of protection, the redo data needed to recover each transaction must be written to both the local online redo log and to the standby redo log on at least one standby database before the transaction commits. To ensure data loss cannot occur, the primary database shuts down if a fault prevents it from writing its redo stream to at least one remote standby redo log. For multiple-instance RAC databases, Data Guard shuts down the primary database if it is unable to write the redo records to at least one properly configured database instance. The maximum protection mode requires that at least one standby instance has a standby redo log and the `LGWR`, `SYNC`, and `AFFIRM` attributes be used on the `LOG_ARCHIVE_DEST_n` parameter for this destination.

5.6.1.2 Maximum Availability Mode

This protection mode provides the highest level of data protection that is possible without compromising the availability of the primary database. Like maximum protection mode, a transaction will not commit until the redo needed to recover that transaction is written to the local online redo log and to at least one remote standby redo log. Unlike maximum protection mode, the primary database does not shut down if a fault prevents it from writing its redo stream to a remote standby redo log. Instead, the primary database operates in maximum performance mode until the fault is corrected and all gaps in redo log files are resolved. When all gaps are resolved, the primary database automatically resumes operating in maximum availability mode.

This mode ensures that no data loss will occur if the primary database fails, but only if a second fault does not prevent a complete set of redo data from being sent from the primary database to at least one standby database.

Like maximum protection mode, the maximum availability mode requires that you:

- Configure standby redo log files on at least one standby database.
- Set the `SYNC`, `LGWR`, and `AFFIRM` attributes of the `LOG_ARCHIVE_DEST_n` parameter for at least 1 standby database.

5.6.1.3 Maximum Performance Mode

This protection mode (the default) provides the highest level of data protection that is possible without affecting the performance of the primary database. This is accomplished by allowing a transaction to commit as soon as the redo data needed to recover that transaction is written to the local online redo log. The primary database's redo data stream is also written to at least one standby database, but that redo stream is written asynchronously with respect to the commitment of the transactions that create the redo data.

When network links with sufficient bandwidth are used, this mode provides a level of data protection that approaches that of maximum availability mode with minimal impact on primary database performance.

The maximum performance mode enables you to either set the `LGWR` and `ASYN` attributes, or set the `ARCH` attribute on the `LOG_ARCHIVE_DEST_n` parameter for the standby database destination. If the primary database fails, you can reduce the amount of data that is not received on the standby destination by setting the `LGWR` and `ASYN` attributes.

5.6.2 Setting the Data Protection Mode of a Data Guard Configuration

To set up redo transport services and specify a level of data protection for the Data Guard configuration, perform the following steps.

Step 1 Configure the `LOG_ARCHIVE_DEST_n` parameters on the primary database.

On the primary database, configure the `LOG_ARCHIVE_DEST_n` parameter attributes appropriately. Each of the Data Guard data protection modes requires that *at least* one standby database in the configuration meet the minimum set of requirements listed in [Table 5–2](#).

Table 5–2 Minimum Requirements for Data Protection Modes

	Maximum Protection	Maximum Availability	Maximum Performance
Redo archival process	LGWR	LGWR	LGWR or ARCH
Network transmission mode	SYNC	SYNC	SYNC or ASYN when using LGWR process. SYNC if using ARCH process
Disk write option	AFFIRM	AFFIRM	AFFIRM or NOAFFIRM
Standby redo log required?	Yes	Yes	No, but it is recommended

Note: Oracle recommends that a Data Guard configuration that is running in maximum protection mode contains *at least* two standby databases that meet the requirements listed in [Table 5–2](#). That way, the primary database can continue processing if one of the standby databases cannot receive redo data from the primary database.

The following example shows how to configure the maximum availability mode:

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_2='SERVICE=chicago
2> OPTIONAL LGWR SYNC AFFIRM
3> VALID_FOR=(ONLINE_LOGFILES, PRIMARY_ROLE)
4> DB_UNIQUE_NAME=chicago';
```

If they are not already specified in the SPFILE, you should also specify unique names with the DB_UNIQUE_NAME initialization parameter and list all databases on the LOG_ARCHIVE_CONFIG parameter with the DG_CONFIG attribute. For example:

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_CONFIG='DG_CONFIG=(chicago,boston)';
```

This will enable the dynamic addition of a standby database to a Data Guard configuration that has a Real Application Clusters primary database running in either maximum protection or maximum availability mode.

Step 1 If you are upgrading the protection mode, perform this step.

Perform this step *only* if you are upgrading the protection mode (for example, from maximum performance to maximum availability mode). Otherwise, go to Step 3.

Assume this example is upgrading the Data Guard configuration from the maximum performance mode to the maximum availability mode. Shut down the primary database and restart it in mounted mode:

```
SQL> SHUTDOWN IMMEDIATE;
SQL> STARTUP MOUNT;
```

For a Real Application Clusters database, shut down all of the primary instances but start and mount only one primary instance.

Step 2 Set the data protection mode.

To specify a data protection mode, issue the SQL ALTER DATABASE SET STANDBY DATABASE TO MAXIMIZE {PROTECTION | AVAILABILITY | PERFORMANCE} statement on the primary database. For example, the following statement specifies the maximum availability mode:

```
SQL> ALTER DATABASE SET STANDBY DATABASE TO MAXIMIZE AVAILABILITY;
```

Step 3 Open the primary database.

If you performed Step 1 to upgrade the protection mode, open the database:

```
SQL> ALTER DATABASE OPEN;
```

If you are downgrading the protection mode, the database will already be open.

Step 4 Configure the LOG_ARCHIVE_DEST_ *n* parameters on standby databases.

On the standby databases, configure the LOG_ARCHIVE_DEST_ *n* parameter attributes so the configuration can continue to operate in the new protection mode after a switchover.

For example:

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_2='SERVICE=boston
2> OPTIONAL LGWR SYNC AFFIRM
3> VALID_FOR=(ONLINE_LOGFILES, PRIMARY_ROLE)
4> DB_UNIQUE_NAME=boston';
```

Step 5 Confirm the configuration is operating in the new protection mode.

Query the V\$DATABASE view to confirm the Data Guard configuration is operating in the new protection mode. For example:

```
SQL> SELECT PROTECTION_MODE, PROTECTION_LEVEL FROM V$DATABASE;
```

PROTECTION_MODE	PROTECTION_LEVEL
-----	-----
MAXIMUM AVAILABILITY	MAXIMUM AVAILABILITY

See [Chapter 15](#) and *Oracle Database SQL Reference* for information about SQL statements.

5.7 Managing Log Files

This section contains the following topics:

- [Specifying Alternate Directory Locations for Archived Redo Log Files](#)
- [Reusing Online Redo Log Files](#)
- [Managing Standby Redo Log Files](#)
- [Planning for Growth and Reuse of the Control Files](#)
- [Sharing a Log File Destination Among Multiple Standby Databases](#)

5.7.1 Specifying Alternate Directory Locations for Archived Redo Log Files

Typically, when redo data is received from the primary database, the redo data is written to archived redo log files that are stored in the directory you specify with the LOCATION attribute of the LOG_ARCHIVE_DEST_ *n* parameter. Alternatively, you can specify the STANDBY_ARCHIVE_DEST initialization parameter on the standby database to indicate an alternate directory where the archived redo log files are to be stored when received from the primary database.

If both parameters are specified, the STANDBY_ARCHIVE_DEST initialization parameter overrides the directory location specified with the LOG_ARCHIVE_DEST_ *n* parameter.

The location where archived redo log files are stored on the standby database is determined according to the following list of rules. When the database instance is started, the archived redo log files are evaluated in the list order:

1. If the STANDBY_ARCHIVE_DEST initialization parameter is specified on the standby database, that location is used.
2. If the LOG_ARCHIVE_DEST_ *n* parameter contains the VALID_FOR= (STANDBY_LOGFILE, *) attribute, then the location specified for this destination is used.
3. If the COMPATIBLE parameter is set to 10.0 or greater and none of the LOG_ARCHIVE_DEST_ *n* parameters contain the VALID_FOR= (STANDBY_LOGFILE, *) attribute, then an arbitrary LOG_ARCHIVE_DEST_ *n* parameter that is valid for the destination is used.
4. If none of the initialization parameters have been specified, then archived redo log files are stored in the default location for the STANDBY_ARCHIVE_DEST initialization parameter.

To see the implicit default value of the STANDBY_ARCHIVE_DEST initialization parameter, query the V\$ARCHIVE_DEST view:

```
SQL> SELECT DEST_NAME, DESTINATION FROM V$ARCHIVE_DEST
2> WHERE DEST_NAME='STANDBY_ARCHIVE_DEST';
```

```
DEST_NAME
```

```
DESTINATION
```

```
STANDBY_ARCHIVE_DEST
/oracle/dbs/arch
```

Redo transport services use the value specified with the `STANDBY_ARCHIVE_DEST` initialization parameter in conjunction with the `LOG_ARCHIVE_FORMAT` parameter to generate the filenames for the archived redo log files on the standby site. For example:

```
STANDBY_ARCHIVE_DEST='/arc_dest/arls'
LOG_ARCHIVE_FORMAT=log%t_%s_%r.arc
```

In the example, `%s` corresponds to the sequence number, and `%r` corresponds to the resetlogs ID. Together, these ensure unique names are constructed for the archived redo log files across multiple incarnations of the database. The `%t`, which is required for Real Application Clusters configurations, corresponds to the thread number.

For a physical standby database, redo transport services store the fully qualified filenames in the standby database control file, and Redo Apply uses this information to perform recovery on the standby database.

Note: If you have specified the `TEMPLATE` attribute of the `LOG_ARCHIVE_DEST_n` parameter, it will override the filename generated with the `STANDBY_ARCHIVE_DEST` and `LOG_ARCHIVE_FORMAT` parameter. See [Chapter 14](#) for information about the `TEMPLATE` attributes.

To display the list of archived redo log files that are on the standby system, query the `V$ARCHIVED_LOG` view on the standby database:

```
SQL> SELECT NAME FROM V$ARCHIVED_LOG;
NAME
```

```
-----
/arc_dest/log_1_771.arc
/arc_dest/log_1_772.arc
/arc_dest/log_1_773.arc
/arc_dest/log_1_774.arc
/arc_dest/log_1_775.arc
```

5.7.2 Reusing Online Redo Log Files

You can specify a policy for reusing the online redo log file by setting the `OPTIONAL` or `MANDATORY` attribute of the `LOG_ARCHIVE_DEST_n` parameter. By default, remote destinations are set to `OPTIONAL`. The archival operation of an optional destination can fail, and the online redo log file can be reused even though transmitting the redo data and writing the log contents was not successful. If the archival operation of a mandatory destination fails, online redo log files cannot be overwritten until the failed archive is completed to the mandatory destination.

By default, one local destination is mandatory even if you designate all destinations to be optional.

[Example 5-8](#) shows how to set a mandatory local archiving destination and enable that destination. When specifying the `MANDATORY` attribute, also consider specifying the `REOPEN` and `MAX_FAILURE` attributes as described in [Section 5.5](#) to handle failure conditions.

Example 5-8 Setting a Mandatory Archiving Destination

```
LOG_ARCHIVE_DEST_3 = 'LOCATION=/arc_dest MANDATORY'
```

5.7.3 Managing Standby Redo Log Files

This section contains the following topics:

- [Determining If a Standby Redo Log File Group Configuration Is Adequate](#)
- [Adding Standby Redo Log Members to an Existing Group](#)

5.7.3.1 Determining If a Standby Redo Log File Group Configuration Is Adequate

The easiest way to verify the standby redo log has an appropriate number of log file groups is to examine the RFS process trace file and database alert log. If either log contains messages that indicate the RFS process frequently has to wait for a group because archiving did not complete, then add more log file groups to the standby redo log. The additional standby redo log file groups give the archival operation time to complete before the standby redo log file is reused by the RFS process.

Caution: Whenever you add an online redo log file group to the primary database, you must add a corresponding standby redo log file group to the standby database. If the number of standby redo log file groups is inadequate, the primary database will shut down if it is operating in maximum protection mode or switch to maximum performance mode if it is operating in maximum availability mode.

5.7.3.2 Adding Standby Redo Log Members to an Existing Group

In some cases, it might not be necessary to create a complete group of standby redo log files. A group could already exist, but may not be complete because one or more members were dropped (for example, because of disk failure). In this case, you can add new members to an existing group.

To add new members to a standby redo log file group, use the `ALTER DATABASE` statement with the `ADD STANDBY LOGFILE MEMBER` clause. The following statement adds a new member to the standby redo log file group number 2:

```
SQL> ALTER DATABASE ADD STANDBY LOGFILE MEMBER '/disk1/oracle/dbs/log2b.rdo'
2> TO GROUP 2;
```

Use fully qualified filenames of new members to indicate where the file should be created. Otherwise, files will be created in either the default or current directory of the database, depending on your operating system.

5.7.4 Planning for Growth and Reuse of the Control Files

This section describes:

- [Sizing the Disk Volumes that Contain the Control Files](#)
- [Specifying the Reuse of Records in the Control File](#)

5.7.4.1 Sizing the Disk Volumes that Contain the Control Files

As archived redo log files are generated and RMAN backups are made, Oracle adds new records to the reusable section of the control file. If no records are available for reuse (because all records are still within the number of days specified by `CONTROL_FILE_RECORD_KEEP_TIME`), then the control file is expanded and new records are added to the control file.

The maximum control file size is 20000 database blocks. If `DB_BLOCK_SIZE` equals 8192, then the maximum control file size is 156 MB. If the control files are stored in pre-created volumes, then the volumes that contain the primary and standby control files should be sized to accommodate a control file of maximum size. If the control file volume is too small and cannot be extended, then existing records in the control file will be overwritten before their intended reuse. This behavior is indicated by the following message in the alert log:

```
krcpwnc: following controlfile record written over:
```

5.7.4.2 Specifying the Reuse of Records in the Control File

The `CONTROL_FILE_RECORD_KEEP_TIME` initialization parameter specifies the minimum number of days that must pass before a reusable record in the control file can be reused. Setting this parameter appropriately prevents redo transport services from overwriting a reusable record in the control file and ensures redo information remains available on the standby database:

- Set `CONTROL_FILE_RECORD_KEEP_TIME` to a value that allows all on disk backup information to be retained in the control file. `CONTROL_FILE_RECORD_KEEP_TIME` specifies the number of days that records are kept within the control file before becoming a candidate for reuse.
- Set `CONTROL_FILE_RECORD_KEEP_TIME` to a value slightly longer than the oldest backup file that you intend to keep on disk, as determined by the size of the backup area.

For example, if the backup area is sized to maintain two full backups that are taken every 7 days, as well as daily incremental backups and archived redo log files, then set `CONTROL_FILE_RECORD_KEEP_TIME` to a value of 21 or 30.

Records older than this will be reused. However, the backup metadata will still be available in the RMAN recovery catalog.

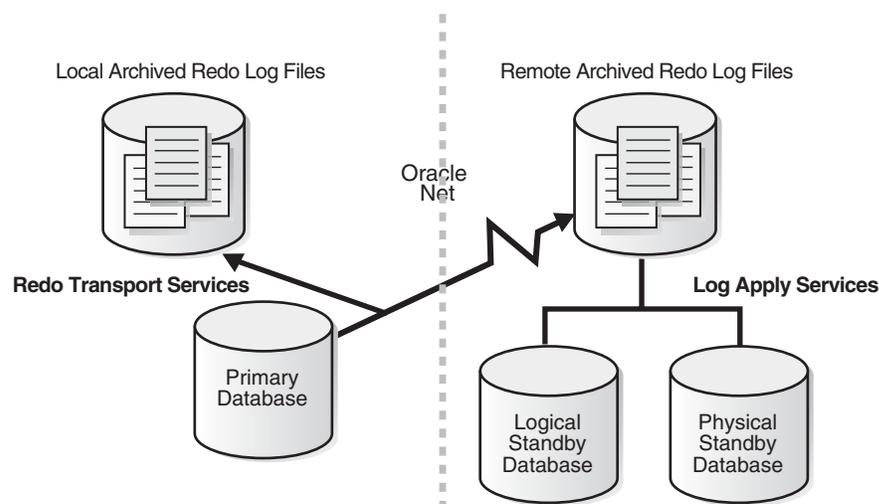
Make sure you specify a large enough value if an apply delay is also set for the standby database (described in [Section 6.2.2](#)). The range of values for this parameter is 0 to 365 days. The default value is 7 days.

See *Oracle Database Reference* for more details about the `CONTROL_FILE_RECORD_KEEP_TIME` initialization parameter and *Oracle Database Backup and Recovery Advanced User's Guide*.

5.7.5 Sharing a Log File Destination Among Multiple Standby Databases

Use the `DEPENDENCY` attribute of the `LOG_ARCHIVE_DEST_n` initialization parameter to define one archival destination to receive redo data on behalf of several destinations, rather than transmitting redo data to each individual destination.

[Figure 5–7](#) shows a Data Guard configuration in which the primary database transports redo data to one archiving destination that shares its archived redo log files with both a logical standby database and a physical standby database. These destinations are dependent on the successful completion of archival operations to the *parent* destination.

Figure 5–7 Data Guard Configuration with Dependent Destinations

Specifying a destination dependency can be useful in the following situations:

- When you configure a physical standby database and a logical standby database on the same system.
- When you configure the standby database and the primary database on the same system. Therefore, the archived redo log files are implicitly accessible to the standby database.
- When clustered file systems are used to provide remote standby databases with access to the primary database archived redo log files.
- When operating system-specific network file systems are used, providing remote standby databases with access to the primary database archived redo log files.

For example, assume there are two standby databases `stbby1` and `stbby2` that reside on the same piece of hardware. There is no need to use network bandwidth and disk space to send the same redo data to both destinations. The databases can share the same archived redo log files if you use the `DEPENDENCY` attribute to designate one of the destinations as being a dependent destination. That is, the primary database sends redo to be archived on the destination that is not defined as the *dependent* destination. If the redo data successfully arrives at that destination, the primary database considers it archived to both destinations. For example:

```
LOG_ARCHIVE_DEST_1='LOCATION=DISK1 MANDATORY'
LOG_ARCHIVE_DEST_2='SERVICE=stbby1 OPTIONAL'
LOG_ARCHIVE_DEST_3='SERVICE=stbby2 OPTIONAL DEPENDENCY=LOG_ARCHIVE_DEST_2'
```

With these parameter definitions, the primary database transmits redo data to `stbby1` but not to `stbby2`. The `stbby2` database instead recovers redo from the archived redo log files that are shipped to `stbby1`.

See Also: The [DEPENDENCY](#) attribute on page 14-10

5.8 Managing Archive Gaps

An **archive gap** can occur on the standby system when it has not received one or more archived redo log files generated by the primary database. The missing archived redo log files are the gap. If there is a gap, it is automatically detected and resolved by Data Guard by copying the missing sequence of log files to the standby destination.

For example, an archive gap can occur when the network becomes unavailable and automatic archiving from the primary database to the standby database temporarily stops. When the network is available again, automatic transmission of the redo data from the primary database to the failed standby database resumes.

Data Guard requires no manual intervention by the DBA to detect and resolve such gaps. The following sections describe gap detection and resolution.

5.8.1 When Is an Archive Gap Discovered?

An archive gap can occur whenever the primary database archives a log locally, but the log is not received at the standby site. Every minute, the primary database polls its standby databases to see if there are gaps in the sequence of archived redo log files.

5.8.2 How Is a Gap Resolved?

Gap recovery is handled through the polling mechanism. For physical and logical standby databases, Oracle Change Data Capture, and Oracle Streams, Data Guard performs gap detection and resolution by automatically retrieving missing archived redo log files from the primary database. No extra configuration settings are required to poll the standby databases, to detect any gaps, or to resolve the gaps.

The important consideration here is that automatic gap recovery is contingent on the availability of the *primary database*. If the primary database is not available and you have a configuration with multiple physical standby databases, you can set up additional initialization parameters so that the Redo Apply can resolve archive gaps from another standby database, as described in [Section 5.8.3](#).

See [Section 12.11](#) for a scenario that shows how to resolve a gap manually.

Note: Prior to Oracle Database 10g Release 1, the FAL client and server were used to resolve gaps from the primary database.

5.8.3 Using the Fetch Archive Log (FAL) to Resolve Archive Gaps

The fetch archive log (FAL) client and server resolve gaps detected in the range of archived redo log files generated at the primary database and received at the physical standby database.

- The FAL client requests the transfer of archived redo log files automatically.
- The FAL server services the FAL requests coming from the FAL client.

The FAL mechanism handles the following types of archive gaps and problems:

- When creating a physical or logical standby database, the FAL mechanism can automatically retrieve any archived redo log files generated during a hot backup of the primary database.
- When there are problems with archived redo log files that have already been received on the standby database, the FAL mechanism can automatically retrieve archived redo log files to resolve any of the following situations:
 - When the archived redo log file is deleted from disk before it is applied to the standby database.
 - When the archived redo log file cannot be applied because of a disk corruption.

- When the archived redo log file is accidentally replaced by another file (for example, a text file) that is not an archived redo log file before the redo data has been applied to the standby database.
- When you have multiple physical standby databases, the FAL mechanism can automatically retrieve missing archived redo log files from another physical standby database.

The FAL client and server are configured using the `FAL_CLIENT` and `FAL_SERVER` initialization parameters that are set on the standby database. Define the `FAL_CLIENT` and `FAL_SERVER` initialization parameters only for physical standby databases in the initialization parameter file as shown in the following table:

Parameter	Function	Syntax
<code>FAL_SERVER</code>	Specifies the network service name that the standby database should use to connect to the FAL server. It can consist of multiple values in a list.	Syntax <code>FAL_SERVER=net_service_name</code> Example <code>FAL_SERVER=standby2_db, standby3_db</code>
<code>FAL_CLIENT</code>	Specifies the network service name that the FAL server should use to connect to the standby database.	Syntax <code>FAL_CLIENT=net_service_name</code> Example <code>FAL_CLIENT=standby1_db</code>

5.8.4 Manually Determining and Resolving Archive Gaps

In some situations, automatic gap recovery may not take place and you will need to perform gap recovery manually. For example, you will need to perform gap recovery manually if you are using logical standby databases and the primary database is not available.

The following sections describe how to query the appropriate views to determine which log files are missing and perform manual recovery.

On a physical standby database

To determine if there is an archive gap on your physical standby database, query the `V$ARCHIVE_GAP` view as shown in the following example:

```
SQL> SELECT * FROM V$ARCHIVE_GAP;
      THREAD#  LOW_SEQUENCE#  HIGH_SEQUENCE#
-----
           1             7             10
```

The output from the previous example indicates your physical standby database is currently missing log files from sequence 7 to sequence 10 for thread 1. After you identify the gap, issue the following SQL statement on the primary database to locate the archived redo log files on your primary database (assuming the local archive destination on the primary database is `LOG_ARCHIVE_DEST_1`):

```
SQL> SELECT NAME FROM V$ARCHIVED_LOG WHERE THREAD#=1 AND DEST_ID=1 AND
2> SEQUENCE# BETWEEN 7 AND 10;
```

```
NAME
-----
/primary/thread1_dest/arcr_1_7.arc
/primary/thread1_dest/arcr_1_8.arc
```

```
/primary/thread1_dest/arcr_1_9.arc
```

Copy these log files to your physical standby database and register them using the ALTER DATABASE REGISTER LOGFILE statement on your physical standby database. For example:

```
SQL> ALTER DATABASE REGISTER LOGFILE
'/physical_standby1/thread1_dest/arcr_1_7.arc';
SQL> ALTER DATABASE REGISTER LOGFILE
'/physical_standby1/thread1_dest/arcr_1_8.arc';
```

After you register these log files on the physical standby database, you can restart Redo Apply.

Note: The V\$ARCHIVE_GAP fixed view on a physical standby database only returns the next gap that is currently blocking Redo Apply from continuing. After resolving the gap and starting Redo Apply, query the V\$ARCHIVE_GAP fixed view again on the physical standby database to determine the next gap sequence, if there is one. Repeat this process until there are no more gaps.

On a logical standby database:

To determine if there is an archive gap, query the DBA_LOGSTDBY_LOG view on the logical standby database. For example, the following query indicates there is a gap in the sequence of archived redo log files because it displays two files for THREAD 1 on the logical standby database. (If there are no gaps, the query will show only one file for each thread.) The output shows that the highest registered file is sequence number 10, but there is a gap at the file shown as sequence number 6:

```
SQL> COLUMN FILE_NAME FORMAT a55
SQL> SELECT THREAD#, SEQUENCE#, FILE_NAME FROM DBA_LOGSTDBY_LOG L
2> WHERE NEXT_CHANGE# NOT IN
3> (SELECT FIRST_CHANGE# FROM DBA_LOGSTDBY_LOG WHERE L.THREAD# = THREAD#)
4> ORDER BY THREAD#,SEQUENCE#;
```

THREAD#	SEQUENCE#	FILE_NAME
1	6	/disk1/oracle/dbs/log-1292880008_6.arc
1	10	/disk1/oracle/dbs/log-1292880008_10.arc

Copy the missing log files, with sequence numbers 7, 8, and 9, to the logical standby system and register them using the ALTER DATABASE REGISTER LOGICAL LOGFILE statement on your logical standby database.

For example:

```
SQL> ALTER DATABASE REGISTER LOGICAL LOGFILE '/disk1/oracle/dbs/log-1292880008_10.arc';
```

After you register these log files on the logical standby database, you can restart SQL Apply.

Note: The DBA_LOGSTDBY_LOG view on a logical standby database only returns the next gap that is currently blocking SQL Apply from continuing. After resolving the identified gap and starting SQL Apply, query the DBA_LOGSTDBY_LOG view again on the logical standby database to determine the next gap sequence, if there is one. Repeat this process until there are no more gaps.

5.9 Verification

This section contains the following topics:

- [Monitoring Log File Archival Information](#)
- [Monitoring the Performance of Redo Transport Services](#)

5.9.1 Monitoring Log File Archival Information

This section describes using views to monitor redo log archival activity for the primary database. See *Oracle Data Guard Broker* and Oracle Enterprise Manager online help for more information about the graphical user interface that automates many of the tasks involved in monitoring a Data Guard environment

Step 1 Determine the status of redo log files.

Enter the following query on the primary database to determine the status of all online redo log files:

```
SQL> SELECT THREAD#, SEQUENCE#, ARCHIVED, STATUS FROM V$LOG;
```

Step 2 Determine the most recent archived redo log file.

Enter the following query on the primary database to determine recently archived thread and sequence number:

```
SQL> SELECT MAX(SEQUENCE#), THREAD# FROM V$ARCHIVED_LOG GROUP BY THREAD#;
```

Step 3 Determine the most recent archived redo log file at each destination.

Enter the following query on the primary database to determine which archived redo log file was most recently transmitted to each of the archiving destinations:

```
SQL> SELECT DESTINATION, STATUS, ARCHIVED_THREAD#, ARCHIVED_SEQ#
2> FROM V$ARCHIVE_DEST_STATUS
3> WHERE STATUS <> 'DEFERRED' AND STATUS <> 'INACTIVE';
```

DESTINATION	STATUS	ARCHIVED_THREAD#	ARCHIVED_SEQ#
/private1/prmy/lad	VALID	1	947
standby1	VALID	1	947

The most recently written archived redo log file should be the same for each archive destination listed. If it is not, a status other than `VALID` might identify an error encountered during the archival operation to that destination.

Step 4 Find out if archived redo log files have been received.

You can issue a query at the primary database to find out if an archived redo log file was not received at a particular site. Each destination has an ID number associated with it. You can query the `DEST_ID` column of the `V$ARCHIVE_DEST` fixed view on the primary database to identify each destination's ID number.

Assume the current local destination is 1, and one of the remote standby destination IDs is 2. To identify which log files are missing at the standby destination, issue the following query:

```
SQL> SELECT LOCAL.THREAD#, LOCAL.SEQUENCE# FROM
2> (SELECT THREAD#, SEQUENCE# FROM V$ARCHIVED_LOG WHERE DEST_ID=1)
3> LOCAL WHERE
4> LOCAL.SEQUENCE# NOT IN
5> (SELECT SEQUENCE# FROM V$ARCHIVED_LOG WHERE DEST_ID=2 AND
```

```
6> THREAD# = LOCAL.THREAD#);
```

```

THREAD# SEQUENCE#
-----
1       12
1       13
1       14

```

See [Appendix A](#) for details about monitoring the archiving status of the primary database.

Step 5 Trace the progression of transmitted redo on the standby site.

To see the progression of the transmission of redo data to the standby destination, set the LOG_ARCHIVE_TRACE parameter in the primary and standby initialization parameter files. See [Appendix G](#) for complete details and examples.

5.9.2 Monitoring the Performance of Redo Transport Services

This section describes the wait events that monitor the performance of the redo transport services that were specified on the primary database with the ARCH, LGWR, SYNC, and ASYNC attributes on the LOG_ARCHIVE_DEST_n initialization parameter.

The following sections describe the wait events and associated timing information that are displayed by the V\$SYSTEM_EVENT view:

- [ARCn Process Wait Events](#)
- [LGWR SYNC Wait Events](#)
- [LGWR ASYNC Wait Events](#)

5.9.2.1 ARCn Process Wait Events

For ARCn archival processing, [Table 5–3](#) shows several of the wait events that monitor the time it takes to spawn or delete RFS connections and to send the redo data to the standby database when using ARCH for the transport mode. See [Section 5.3.1](#) for information about ARCn archival processing.

Table 5–3 Wait Events for Destinations Configured with the ARCH Attribute

Wait Event	Monitors the Amount of Time Spent By . . .
ARCH wait on ATTACH	All ARCn processes to spawn an RFS connection.
ARCH wait on SENDREQ	All ARCn processes to write the received redo data to disk as well as open and close the remote archived redo log files.
ARCH wait on DETACH	All ARCn processes to delete an RFS connection.

5.9.2.2 LGWR SYNC Wait Events

For LGWR SYNC archival processing, [Table 5–4](#) shows several of the wait events that monitor the time it takes for the LGWR process on the primary database to:

- Complete writing to the online redo log files on the primary database
- Transmit the redo data to the remote standby destination
- Wait for the redo data to be written to the standby redo log files
- Receive acknowledgment from the remote standby destination

See [Section 5.3.2](#) for information about LGWR SYNC archival processing.

Table 5–4 Wait Events for Destinations Configured with the LGWR SYNC Attributes

Wait Event	Monitors the Amount of Time Spent By . . .
LGWR wait on LNS	The LGWR process waiting to receive messages from the LNS n process.
LNS wait on ATTACH	All network servers to spawn an RFS connection.
LNS wait on SENDREQ	All network servers to write the received redo data to disk as well as open and close the remote archived redo log files.
LNS wait on DETACH	All network servers to delete an RFS connection.

5.9.2.3 LGWR ASYNC Wait Events

For LGWR ASYNC archival processing, [Table 5–5](#) shows several of the wait events that monitor the time it takes to write the redo data to the online redo log files on the primary database. See [Section 5.3.2](#) for information about LGWR ASYNC archival processing.

Table 5–5 Wait Events for Destinations Configured with the LGWR ASYNC Attributes

Wait Event	Monitors the Amount of Time Spent By . . .
LNS wait on DETACH	All network servers to delete an RFS connection.
LNS wait on ATTACH	All network servers to spawn an RFS connection.
LNS wait on SENDREQ	All network servers to write the received redo data to disk as well as open and close the remote archived redo log files.
True ASYNC Control FileTXN Wait	The LNS n process to get hold of the control file transaction during its lifetime.
True ASYNC Wait for ARCH log	The LNS n process waiting to see the archived redo log (if the LNS n process is archiving a current log file and the log is switched out).
Waiting for ASYNC dest activation	The LNS n process waiting for an inactive destination to become active.
True ASYNC log-end-of-file wait	The LNS n process waiting for the next bit of redo after it has reached the logical end of file.

Log Apply Services

This chapter describes how redo data is applied to a standby database. It includes the following topics:

- [Introduction to Log Apply Services](#)
- [Log Apply Services Configuration Options](#)
- [Applying Redo Data to Physical Standby Databases](#)
- [Applying Redo Data to Logical Standby Databases](#)

6.1 Introduction to Log Apply Services

Log apply services automatically apply *redo* to standby databases to maintain synchronization with the primary database and allow transactionally consistent access to the data.

By default, log apply services wait for the *full* archived redo log file to arrive on the standby database before applying it to the standby database. [Section 5.3.1](#) and [Section 5.3.2](#) describe how redo data transmitted from the primary database is received by the remote file server process (RFS) on the standby system where the RFS process writes the redo data to either archived redo log files or standby redo log files. However, if you use standby redo log files, you can enable **real-time apply**, which allows Data Guard to recover redo data from the current standby redo log file as it is being filled up by the RFS process. Real-time apply is described in more detail in [Section 6.2.1](#).

Log apply services use the following methods to maintain physical and logical standby databases:

- Redo apply (physical standby databases only)
Uses media recovery to keep the primary and physical standby databases synchronized.

Caution: You can also open a physical standby database in read-only mode to allow users to query the standby database for reporting purposes. While open, redo data is still received; however, Redo Apply stops and the physical standby database is not kept synchronized with the primary database. If a failure occurs during this time, it can prolong the time it takes for a failover operation to complete. See [Section 8.2, "Opening a Standby Database for Read-Only or Read/Write Access"](#) for more information.

- **SQL Apply (logical standby databases only)**
Reconstitutes SQL statements from the redo received from the primary database and executes the SQL statements against the logical standby database.

Logical standby databases can be opened in read/write mode, but the target tables being maintained by the logical standby database are opened in read-only mode for reporting purposes (providing the database guard was set appropriately). SQL Apply enables you to use the logical standby database for reporting activities, even while SQL statements are being applied.

The sections in this chapter describe Redo Apply, SQL Apply, real-time apply, and delayed apply in more detail.

6.2 Log Apply Services Configuration Options

This section contains the following topics:

- [Using Real-Time Apply to Apply Redo Data Immediately](#)
- [Specifying a Time Delay for the Application of Archived Redo Log Files](#)

6.2.1 Using Real-Time Apply to Apply Redo Data Immediately

If the real-time apply feature is enabled, log apply services can apply redo data as it is received, without waiting for the current standby redo log file to be archived. This results in faster switchover and failover times because the standby redo log files have been applied already to the standby database by the time the failover or switchover begins.

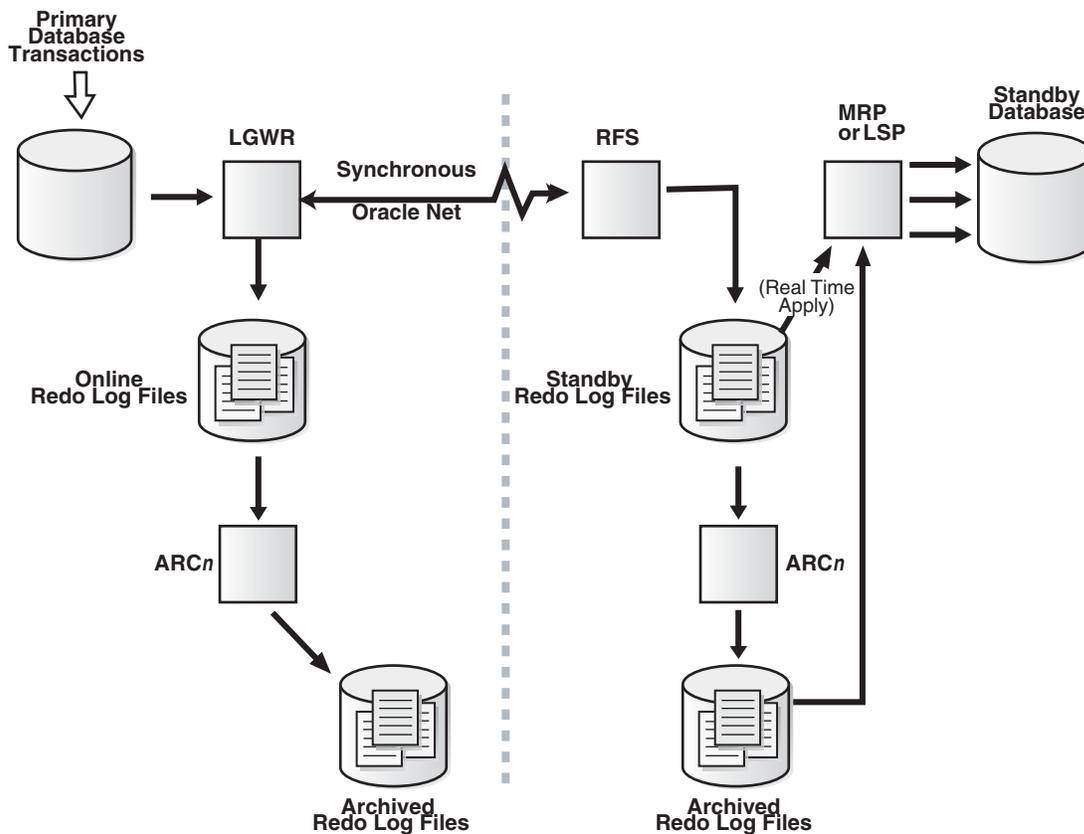
Use the `ALTER DATABASE` statement to enable the real-time apply feature, as follows:

- For physical standby databases, issue the `ALTER DATABASE RECOVER MANAGED STANDBY DATABASE USING CURRENT LOGFILE` statement.
- For logical standby databases, issue the `ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE` statement.

Standby redo log files are required to use real-time apply.

[Figure 6-1](#) shows a Data Guard configuration with a local destination and a standby destination. As the remote file server (RFS) process writes the redo data to standby redo log files on the standby database, log apply services can recover redo from standby redo log files as they are being filled.

Figure 6–1 Applying Redo Data to a Standby Destination Using Real-Time Apply



6.2.2 Specifying a Time Delay for the Application of Archived Redo Log Files

In some cases, you may want to create a time lag between the time when redo data is received from the primary site and when it is applied to the standby database. You can specify a time interval (in minutes) to protect against the application of corrupted or erroneous data to the standby database. When you set a `DELAY` interval, it does not delay the transport of the redo data to the standby database. Instead, the time lag you specify begins when the redo data is completely archived at the standby destination.

Note: If you define a delay for a destination that has real-time apply enabled, the delay is ignored.

Specifying a Time Delay

You can set a time delay on primary and standby databases using the `DELAY=minutes` attribute of the `LOG_ARCHIVE_DEST_n` initialization parameter to delay applying archived redo log files to the standby database. By default, there is no time delay. If you specify the `DELAY` attribute without specifying a value, then the default delay interval is 30 minutes.

Canceling a Time Delay

You can cancel a specified delay interval as follows:

- For physical standby databases, use the `NODELAY` keyword of the `RECOVER MANAGED STANDBY DATABASE` clause:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE NODELAY;
```

- For logical standby databases, specify the following SQL statement:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY NODELAY;
```

These commands result in log apply services immediately beginning to apply archived redo log files to the standby database, before the time interval expires. Also, see:

- [Section 12.8, "Using a Physical Standby Database with a Time Lag"](#)
- *Oracle Database SQL Reference* for the `DELAY` attribute of the `ALTER DATABASE RECOVER MANAGED STANDBY DATABASE` statement

6.2.2.1 Using Flashback Database as an Alternative to Setting a Time Delay

As an alternative to setting an apply delay, you can use Flashback Database to recover from the application of corrupted or erroneous data to the standby database. Flashback Database can quickly and easily flash back a standby database to an arbitrary point in time.

See [Chapter 12](#) for scenarios showing how to use Data Guard with Flashback Database, and *Oracle Database Backup and Recovery Basics* for more information about enabling and using Flashback Database.

6.3 Applying Redo Data to Physical Standby Databases

By default, the redo data is applied from archived redo log files. When performing Redo Apply, a physical standby database can use the real-time apply feature to apply redo directly from the standby redo log files as they are being written by the RFS process. Note that log apply services cannot apply redo data to a physical standby database when it is opened in read-only mode.

This section contains the following topics:

- [Starting Redo Apply](#)
- [Stopping Redo Apply](#)
- [Monitoring Redo Apply on Physical Standby Databases](#)

6.3.1 Starting Redo Apply

To start log apply services on a physical standby database, ensure the physical standby database is started and mounted and then start Redo Apply using the SQL `ALTER DATABASE RECOVER MANAGED STANDBY DATABASE` statement.

You can specify that Redo Apply runs as a foreground session or as a background process, and enable it with real-time apply.

- To start Redo Apply in the foreground, issue the following SQL statement:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE;
```

If you start a foreground session, control is not returned to the command prompt until recovery is canceled by another session.

- To start Redo Apply in the background, include the `DISCONNECT` keyword on the SQL statement. For example:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE DISCONNECT;
```

This statement starts a detached server process and immediately returns control to the user. While the managed recovery process is performing recovery in the background, the foreground process that issued the RECOVER statement can continue performing other tasks. This does not disconnect the current SQL session.

- To start real-time apply, include the USING CURRENT LOGFILE clause on the SQL statement. For example:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE USING CURRENT LOGFILE;
```

6.3.2 Stopping Redo Apply

To stop Redo Apply, issue the following SQL statement in another window:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE CANCEL;
```

6.3.3 Monitoring Redo Apply on Physical Standby Databases

To monitor the status of log apply services on a physical standby database, see [Section 8.5.4](#). You can also monitor the standby database using Oracle Enterprise Manager. Also, see the *Oracle Database Reference* for complete reference information about views.

6.4 Applying Redo Data to Logical Standby Databases

SQL Apply converts the data from the archived redo log or standby redo log in to SQL statements and then executes these SQL statements on the logical standby database. Because the logical standby database remains open, tables that are maintained can be used simultaneously for other tasks such as reporting, summations, and queries.

This section contains the following topics:

- [Starting SQL Apply](#)
- [Stopping SQL Apply on a Logical Standby Database](#)
- [Monitoring SQL Apply on Logical Standby Databases](#)

6.4.1 Starting SQL Apply

To start SQL Apply, start the logical standby database and issue the following statement:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY;
```

To start real-time apply on the logical standby database to immediately apply redo data from the standby redo log files on the logical standby database, include the IMMEDIATE keyword as shown in the following statement:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE;
```

6.4.2 Stopping SQL Apply on a Logical Standby Database

To stop SQL Apply, issue the following statement on the logical standby database:

```
SQL> ALTER DATABASE STOP LOGICAL STANDBY APPLY;
```

When you issue this statement, SQL Apply waits until it has committed all complete transactions that were in the process of being applied. Thus, this command may not stop the SQL Apply processes immediately.

If you want to stop SQL Apply immediately, issue the following statement:

```
SQL> ALTER DATABASE ABORT LOGICAL STANDBY APPLY;
```

6.4.3 Monitoring SQL Apply on Logical Standby Databases

To monitor SQL Apply, see [Section 9.2](#). You can also monitor the standby database using Oracle Enterprise Manager. See [Appendix A, "Troubleshooting Data Guard"](#) and *Oracle Data Guard Broker*.

Also, see the discussion about the `V$ARCHIVE_DEST_STATUS` fixed view in [Section 8.5.4.3](#) and the *Oracle Database Reference* for complete reference information about views.

Role Transitions

A Data Guard configuration consists of one database that functions in the primary role and one or more databases that function in standby roles. Typically, the role of each database does not change. However, if Data Guard is used to maintain service in response to a primary database outage, you must initiate a role transition between the current primary database and one standby database in the configuration. To see the current role of the databases, query the `DATABASE_ROLE` column in the `V$DATABASE` view.

The number, location, and type (physical or logical) of standby databases in the Data Guard configuration and the way in which redo data from the primary database is propagated to each standby database determine the role-management options available to you in response to a primary database outage.

This chapter describes how to manage role transitions in a Data Guard configuration. It contains the following topics:

- [Introduction to Role Transitions](#)
- [Role Transitions Involving Physical Standby Databases](#)
- [Role Transitions Involving Logical Standby Databases](#)
- [Using Flashback Database After a Role Transition](#)

The role transitions described in this chapter are invoked manually using SQL statements. You can also use the Oracle Data Guard broker to simplify role transitions and automate failovers.

See Also: *Oracle Data Guard Broker* for information about using the Oracle Data Guard broker to:

- Simplify switchovers and failovers by allowing you to invoke them using either a single key click in Oracle Enterprise Manager or a single command in the DGMGRL command-line interface.
- Enable **fast-start failover** to fail over *automatically* when the primary database becomes unavailable. When fast-start failover is enabled, the Data Guard broker determines if a failover is necessary and initiates the failover to the specified target standby database automatically, with no need for DBA intervention and with no loss of data.

7.1 Introduction to Role Transitions

A database operates in one of the following mutually exclusive roles: **primary** or **standby**. Data Guard enables you to change these roles dynamically by issuing the

SQL statements described in this chapter, or by using either of the Data Guard broker's interfaces. Oracle Data Guard supports the following role transitions:

- **Switchover**

Allows the primary database to switch roles with one of its standby databases. There is no data loss during a switchover. After a switchover, each database continues to participate in the Data Guard configuration with its new role.

- **Failover**

Changes a standby database to the primary role in response to a primary database failure. If the primary database was not operating in either maximum protection mode or maximum availability mode before the failure, some data loss may occur. If Flashback Database is enabled on both the primary and standby databases, the failed database may be reinstated as a standby for the new primary database once the reason for the failure is corrected.

[Section 7.1.1, "Preparing for a Role Transition \(Failover or Switchover\)"](#) on page 7-2 helps you choose the role transition that best minimizes downtime and risk of data loss. Switchovers and failovers are described in more detail in [Section 7.1.3, "Switchovers"](#) on page 7-4 and [Section 7.1.4, "Failovers"](#) on page 7-6, respectively.

7.1.1 Preparing for a Role Transition (Failover or Switchover)

Before starting any role transition, perform the following preparations:

- Verify the initialization parameters for each database are configured correctly. See [Chapter 3, "Creating a Physical Standby Database"](#) and [Chapter 4, "Creating a Logical Standby Database"](#) for information about how to configure initialization parameters on the primary and standby databases so that the Data Guard configuration operates properly after the role transition.

Also, see [Section 3.1.3, "Configure a Standby Redo Log"](#) on page 3-2 for information about manually adding redo log files when creating a physical standby database.

Note: You must define the `LOG_ARCHIVE_DEST_n` and `LOG_ARCHIVE_DEST_STATE_n` parameters on each standby database so that when a switchover or failover occurs, all standby sites continue to receive redo data from the new primary database. See [Section 5.4.1, "Specifying Role-Based Destinations with the VALID_FOR Attribute"](#) on page 5-14 and [Chapter 14, "LOG_ARCHIVE_DEST_n Parameter Attributes"](#) for information about using the `LOG_ARCHIVE_DEST_n VALID_FOR` attribute to define role-based destinations in preparation for future role transitions.

- Verify the standby database that will become the new primary database is operating in ARCHIVELOG mode.
- Ensure temporary files exist on the standby database that match the temporary files on the primary database.
- Remove any delay in applying redo that may be in effect on the standby database that will become the new primary database.
- Verify that all but one RAC instance on the standby databases in a Real Application Clusters configuration are shut down.

For a Real Application Clusters database, only one RAC instance on the standby database can be online during the role transition. Shut down all other instances before starting the role transition. Then, after the role transition completes, bring these instances back online.

Note: Even though only one RAC instance on the standby database is open during the switchover, all other standby database instances will automatically undergo a transition to their new role correctly when they are opened.

7.1.2 Choosing a Target Standby Database for a Role Transition

For a Data Guard configuration with multiple standby databases, there are a number of factors to consider when choosing the target standby database for a role transition. These include the following:

- Locality of the standby database.
- The capability of the standby database (hardware specifications—such as the number of CPUs, I/O bandwidth available, and so on).
- The time it will take to perform the role transition. This is affected by how far behind the standby database is in terms of application of redo data, and how much flexibility you have in terms of trading off application availability with data loss.

Data Guard provides the `V$DATAGUARD_STATS` view that can be used to estimate the viability of each standby database in terms of the currency of the data in the standby database, and the time it will take to perform a role transition if all available redo data is applied to the standby database. For example:

```
SQL> COLUMN NAME FORMAT A18
SQL> COLUMN VALUE FORMAT A16
SQL> COLUMN TIME_COMPUTED FORMAT A24
SQL> SELECT * FROM V$DATAGUARD_STATS;
NAME                                VALUE                                TIME_COMPUTED
-----                                -
apply finish time                    +00 00:00:02.4                      15-MAY-2005 10:32:49
      second(1)
      interval
apply lag                              +00 0:00:04                          15-MAY-2005 10:32:49
      second(0)
      interval
transport lag                         +00 00:00:00                          15-MAY-2005 10:32:49
      second(0)
      interval
```

This shows that for this standby database, there is no transport lag, that log apply services have not applied the redo generated in the last 4 seconds (`apply lag`), and that it will take log apply services 2.4 seconds to finish applying the unapplied redo (`apply finish time`). The time at which each of the statistics is computed is shown in the `TIME_COMPUTED` column.

If the configuration contains both physical and logical standby databases, consider choosing a physical standby database to be the target standby database. A switchover or failover to a physical standby database is preferable because all databases in the configuration will be viable as standby databases to the new primary database after the role transition completes. Whereas a switchover or failover to a logical standby database will invalidate the other physical standby databases to the original primary

database. You will then need to re-create the physical standby databases from a backup of the new primary database before you can reenble them.

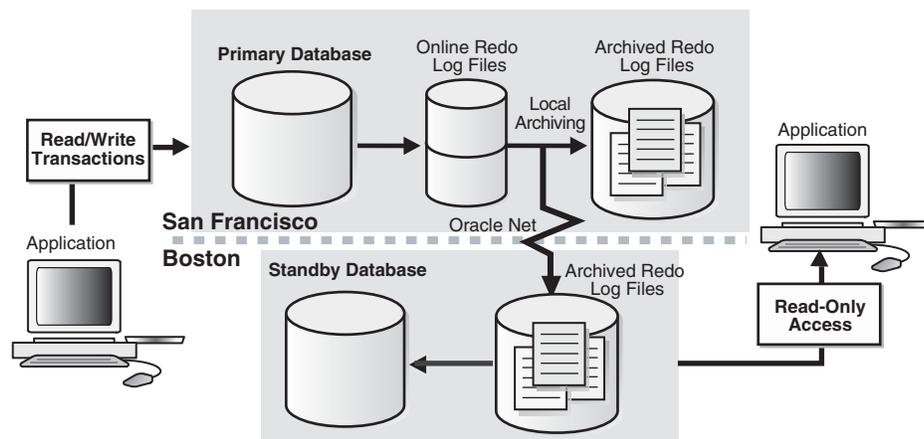
7.1.3 Switchovers

A switchover is typically used to reduce primary database downtime during planned outages, such as operating system or hardware upgrades, or rolling upgrades of the Oracle database software and patch sets (described in [Chapter 11, "Using SQL Apply to Upgrade the Oracle Database"](#)).

A switchover takes place in two phases. In the first phase, the existing primary database undergoes a transition to a standby role. In the second phase, a standby database undergoes a transition to the primary role.

[Figure 7-1](#) shows a two-site Data Guard configuration before the roles of the databases are switched. The primary database is in San Francisco, and the standby database is in Boston.

Figure 7-1 Data Guard Configuration Before Switchover



[Figure 7-2](#) shows the Data Guard environment after the original primary database was switched over to a standby database, but before the original standby database has become the new primary database. At this stage, the Data Guard configuration temporarily has two standby databases.

Figure 7-2 Standby Databases Before Switchover to the New Primary Database

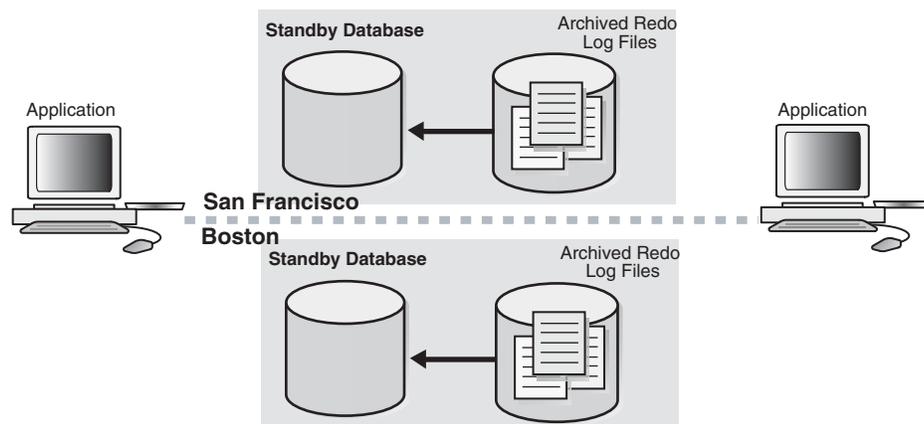
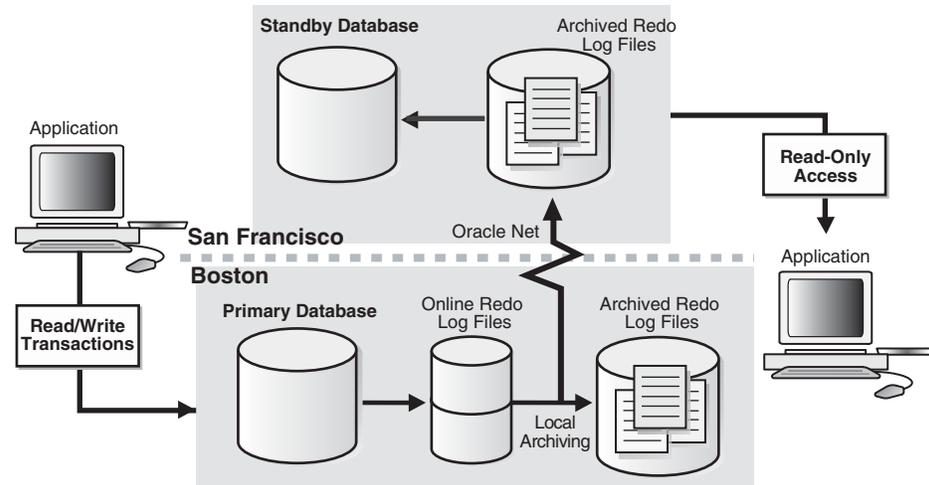


Figure 7-3 shows the Data Guard environment after a switchover took place. The original standby database became the new primary database. The primary database is now in Boston, and the standby database is now in San Francisco.

Figure 7-3 Data Guard Environment After Switchover



Preparing for a Switchover

Ensure the prerequisites listed in [Section 7.1.1](#) are satisfied. In addition, the following prerequisites must be met for a switchover:

- For switchovers involving a physical standby database, verify the primary database instance is open and the standby database instance is mounted.

The standby database that you plan to change to the primary role must be mounted before you begin the switchover. Ideally, the physical standby database will also be actively applying redo when the database roles are switched. If the physical standby database is open for read-only access, the switchover still will take place, but will require additional time. See [Section 6.3, "Applying Redo Data to Physical Standby Databases"](#) on page 6-4 for more information about Redo Apply.
- For switchovers involving a logical standby database, verify both the primary and standby database instances are open and that SQL Apply is active. See [Section 6.4, "Applying Redo Data to Logical Standby Databases"](#) on page 6-5 for more information about SQL Apply.
- For switchovers involving a primary database in a Real Applications Cluster, all but one instance must be shut down. Once the switchover is performed successfully, you can bring all other instances back online.

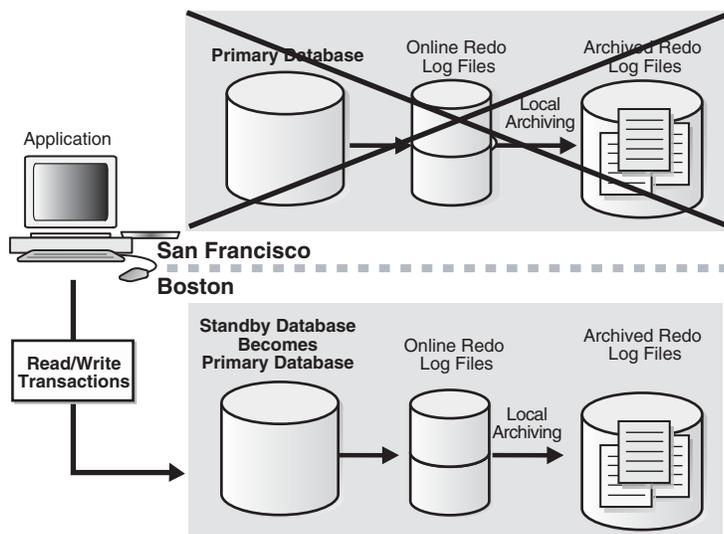
When a database transitions from one role to another, the `DB_ROLE_CHANGE` system event fires. You can write a trigger that's associated with this system event to manage tasks after a switchover occurs. The event fires when the database opens for the first time after the switchover regardless of its new role (that is, regardless of whether the switchover caused it to open for the first time as a primary database, as a logical standby, or as a physical standby in read-only mode). You can query the `DATABASE_ROLE` column of the `V$DATABASE` view to determine a database's current role. See the table of system manager events in *Oracle Database Application Developer's Guide - Fundamentals* for more details.

7.1.4 Failovers

A failover is typically used only when the primary database becomes unavailable, and there is no possibility of restoring it to service within a reasonable period of time. The specific actions performed during a failover vary based on whether a logical or a physical standby database is involved in the failover, the state of the Data Guard configuration at the time of the failover, and on the specific SQL statements used to initiate the failover.

Figure 7-4 shows the result of a failover from a primary database in San Francisco to a physical standby database in Boston.

Figure 7-4 Failover to a Standby Database



Preparing for a Failover

If possible, before performing a failover, you should transfer as much of the available and unapplied primary database redo data as possible to the standby database.

Ensure the prerequisites listed in [Section 7.1.1, "Preparing for a Role Transition \(Failover or Switchover\)"](#) on page 7-2 are satisfied. In addition, the following prerequisites must be met for a failover:

- If a standby database currently running in maximum protection mode will be involved in the failover, first place it in maximum performance mode by issuing the following statement on the standby database:

```
SQL> ALTER DATABASE SET STANDBY DATABASE TO MAXIMIZE PERFORMANCE;
```

Then, if appropriate standby databases are available, you can reset the desired protection mode on the new primary database after the failover completes.

This is required because you cannot fail over to a standby database that is in maximum protection mode. In addition, if a primary database in maximum protection mode is still actively communicating with the standby database, issuing the `ALTER DATABASE` statement to change the standby database from maximum protection mode to maximum performance mode will not succeed. Because a failover removes the original primary database from the Data Guard configuration, these features serve to protect a primary database operating in maximum protection mode from the effects of an unintended failover.

Note: Do not fail over to a standby database to test whether or not the standby database is being updated correctly. Instead:

- See [Section 3.2.7, "Verify the Physical Standby Database Is Performing Properly"](#)
 - See [Section 4.2.6, "Verify the Logical Standby Database Is Performing Properly"](#)
-
-

When a database transitions from one role to another, the `DB_ROLE_CHANGE` system event fires. You can write a trigger that's associated with this system event to manage tasks after a failover occurs. The event fires when the database opens for the first time after the failover regardless of its new role (in the case of a failover to a physical standby database, the system event will fire when the database is opened for the first time after a failover operation). You can query the `DATABASE_ROLE` column of the `V$DATABASE` view to determine a database's current role. See the table of system manager events in *Oracle Database Application Developer's Guide - Fundamentals* for more details.

To perform a failover involving a physical standby database, see [Section 7.2.2, "Failovers Involving a Physical Standby Database"](#) on page 7-9. To perform a failover involving a logical standby database, see [Section 7.3.2, "Failovers Involving a Logical Standby Database"](#) on page 7-16. To perform a failover using the Data Guard broker, see the chapter about "Switchover and Failover Operations" in *Oracle Data Guard Broker*.

7.2 Role Transitions Involving Physical Standby Databases

This section describes how to perform switchovers and failovers involving a physical standby database.

7.2.1 Switchovers Involving a Physical Standby Database

This section describes how to perform a switchover. A switchover must be initiated on the current primary database and completed on the target standby database. The following steps describe how to perform a switchover.

Step 1 Verify it is possible to perform a switchover.

On the current primary database, query the `SWITCHOVER_STATUS` column of the `V$DATABASE` fixed view on the primary database to verify it is possible to perform a switchover. For example:

```
SQL> SELECT SWITCHOVER_STATUS FROM V$DATABASE;
SWITCHOVER_STATUS
-----
TO STANDBY
1 row selected
```

The `TO STANDBY` value in the `SWITCHOVER_STATUS` column indicates that it is possible to switch the primary database to the standby role. If the `TO STANDBY` value is not displayed, then verify the Data Guard configuration is functioning correctly (for example, verify all `LOG_ARCHIVE_DEST_n` parameter values are specified correctly).

If the value in the `SWITCHOVER_STATUS` column is `SESSIONS ACTIVE`, perform the steps described in [Section A.4, "Problems Switching Over to a Standby Database"](#) on page A-4 to identify and terminate active user or SQL sessions that might prevent a

switchover from being processed. If, after performing these steps, the `SWITCHOVER_STATUS` column still displays `SESSIONS ACTIVE`, you can successfully perform a switchover by appending the `WITH SESSION SHUTDOWN` clause to the `ALTER DATABASE COMMIT TO SWITCHOVER TO PHYSICAL STANDBY` statement described in Step 2.

See *Oracle Database Reference* for information about other valid values for the `SWITCHOVER_STATUS` column of the `V$DATABASE` view.

Step 2 Initiate the switchover on the primary database.

To change the current primary database to a physical standby database role, use the following SQL statement on the primary database:

```
SQL> ALTER DATABASE COMMIT TO SWITCHOVER TO PHYSICAL STANDBY;
```

After this statement completes, the primary database is converted into a standby database. The current control file is backed up to the current SQL session trace file before the switchover. This makes it possible to reconstruct a current control file, if necessary.

Step 3 Shut down and restart the former primary instance.

Shut down the former primary instance, and restart and mount the database:

```
SQL> SHUTDOWN IMMEDIATE;  
SQL> STARTUP MOUNT;
```

At this point in the switchover process, both databases are configured as standby databases (see [Figure 7-2](#)).

Step 4 Verify the switchover status in the V\$DATABASE view.

After you change the primary database to the physical standby role and the switchover notification is received by the standby databases in the configuration, you should verify if the switchover notification was processed by the target standby database by querying the `SWITCHOVER_STATUS` column of the `V$DATABASE` fixed view on the target standby database.

For example:

```
SQL> SELECT SWITCHOVER_STATUS FROM V$DATABASE;  
SWITCHOVER_STATUS  
-----  
TO_PRIMARY  
1 row selected
```

If the value in the `SWITCHOVER_STATUS` column is `SESSIONS ACTIVE`, perform the steps described in [Section A.4, "Problems Switching Over to a Standby Database"](#) on page A-4 to identify and terminate active user or SQL sessions that might prevent a switchover from being processed. If, after performing these steps, the `SWITCHOVER_STATUS` column still displays `SESSIONS ACTIVE`, you can proceed to Step 5, and append the `WITH SESSION SHUTDOWN` clause to the switchover statement. See *Oracle Database Reference* for information about other valid values for the `SWITCHOVER_STATUS` column of the `V$DATABASE` view.

Step 5 Switch the target physical standby database role to the primary role.

You can switch a physical standby database from the standby role to the primary role when the standby database instance is either mounted in Redo Apply mode or open for read-only access. It must be in one of these modes so that the primary database

switchover request can be coordinated. After the standby database is in an appropriate mode, issue the following SQL statement on the physical standby database that you want to change to the primary role:

```
SQL> ALTER DATABASE COMMIT TO SWITCHOVER TO PRIMARY;
```

Step 6 Finish the transition of the standby database to the primary role.

The task you perform is dependent on if the physical standby database has ever been opened in read-only mode:

- If the physical standby database *has not been* opened in read-only mode since the last time it was started, issue the SQL `ALTER DATABASE OPEN` statement to open the new primary database:

```
SQL> ALTER DATABASE OPEN;
```

Then, go to step 7 on page 7-9.

- If the physical standby database *has been* opened in read-only mode since the last time it was started, you must shut down the target standby database and restart it:

```
SQL> SHUTDOWN IMMEDIATE;
```

```
SQL> STARTUP;
```

The target physical standby database has now undergone a transition to the primary database role. See [Section 5.4.1, "Specifying Role-Based Destinations with the VALID_FOR Attribute"](#) and [Chapter 14, "LOG_ARCHIVE_DEST_n Parameter Attributes"](#) for information about using the `LOG_ARCHIVE_DEST_n VALID_FOR` attribute to ensure the Data Guard configuration operates properly after a role transition.

Note: There is no need to shut down and restart other standby databases (not involved in the switchover) that are online at the time of the switchover. These standby databases will continue to function normally after the switchover completes.

Step 7 If necessary, restart log apply services on the standby databases.

For the new physical standby database and for each other physical or logical standby database in the Data Guard configuration, if log apply services were not previously configured to continue operating through a switchover, use an appropriate command to restart log apply services. See [Chapter 6, "Log Apply Services"](#) for more information about how to configure and start log apply services.

Step 8 Begin sending redo data to the standby databases.

Issue the following statement on the new primary database:

```
SQL> ALTER SYSTEM SWITCH LOGFILE;
```

7.2.2 Failovers Involving a Physical Standby Database

This section describes how to perform failovers involving a physical standby database.

During failovers involving a physical standby database:

- In all cases, after a failover, the original primary database can no longer participate in the Data Guard configuration.

- In most cases, other logical or physical standby databases not directly participating in the failover remain in the configuration and do not have to be shut down or restarted.
- In some cases, it might be necessary to re-create all standby databases after configuring the new primary database.

These cases are described, where appropriate, within the failover steps below.

Before starting the failover, perform as many of the steps documented in [Section 7.1.4, "Failovers"](#) as possible to prepare the selected standby database for a failover, then proceed to [Section 7.2.2, "Failovers Involving a Physical Standby Database"](#) for the failover steps.

Note: Oracle recommends you use only the failover steps and commands described in the following sections to perform a failover. Do not use the `ALTER DATABASE ACTIVATE STANDBY DATABASE` to perform a failover, because this statement may cause data loss.

Failover Steps

This section describes the steps that must be performed to transition the selected physical standby database to the primary role. Any other physical or logical standby databases that are also part of the configuration will remain in the configuration and will not need to be shut down or restarted.

If the target standby database was operating in maximum protection mode or maximum availability mode using the log writer process (LGWR), no gaps in the archived redo log files should exist, and you can proceed directly to Step 4. Otherwise, begin with Step 1 to determine if any manual gap resolution steps must be performed.

Step 1 Identify and resolve any gaps in the archived redo log files.

To determine if there are gaps in the archived redo log files on the target standby database, query the `V$ARCHIVE_GAP` view.

The `V$ARCHIVE_GAP` view contains the sequence numbers of the archived redo log files that are known to be missing for each thread. The data returned reflects the highest gap only.

For example:

```
SQL> SELECT THREAD#, LOW_SEQUENCE#, HIGH_SEQUENCE# FROM V$ARCHIVE_GAP;
THREAD#      LOW_SEQUENCE# HIGH_SEQUENCE#
-----
           1             90             92
```

In this example the gap comprises archived redo log files with sequences 90, 91, and 92 for thread 1. If possible, copy all of the identified missing archived redo log files to the target standby database from the primary database and register them. This must be done for each thread.

For example:

```
SQL> ALTER DATABASE REGISTER PHYSICAL LOGFILE 'filespec1';
```

Step 2 Repeat Step 1 until all gaps are resolved.

The query executed in Step 1 displays information for the highest gap only. After resolving that gap, you must repeat Step 1 until the query returns no rows.

Step 3 Copy any other missing archived redo log files.

To determine if there are any other missing archived redo log files, query the V\$ARCHIVED_LOG view on the target standby database to obtain the highest sequence number for each thread.

For example:

```
SQL> SELECT UNIQUE THREAD# AS THREAD, MAX(SEQUENCE#)
      2> OVER (PARTITION BY thread#) AS LAST from V$ARCHIVED_LOG;
```

THREAD	LAST
1	100

Copy any available archived redo log files from the primary database that contains sequence numbers higher than the highest sequence number available on the target standby database to the target standby database and register them. This must be done for each thread.

For example:

```
SQL> ALTER DATABASE REGISTER PHYSICAL LOGFILE 'filespec1';
```

After all available archived redo log files have been registered, query the V\$ARCHIVE_GAP view as described in Step 1 to verify no additional gaps were introduced in Step 3.

Note: If, while performing Steps 1 through 3, you are not able to resolve gaps in the archived redo log files (for example, because you do not have access to the system that hosted the failed primary database), some data loss will occur during the failover.

Step 4 Initiate a failover on the target physical standby database.

Issue the following statement to initiate the failover:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE FINISH FORCE;
```

The FORCE keyword terminates active RFS processes on the target physical standby database, so that failover can proceed immediately without waiting for network connections to time out:

Note: Failover adds an end-of-redo marker to the header of the last log file being archived and sends the redo to all enabled destinations that are valid for the primary role (specified with the VALID_FOR= (PRIMARY_ROLE, *_LOGFILES) or the VALID_FOR= (ALL_ROLES, *_LOGFILES) attributes).

Note: The FINISH keyword must follow all other keywords in the SQL statement, except for FORCE, WAIT, or NOWAIT.

Step 5 Convert the physical standby database to the primary role.

Once the SQL ALTER DATABASE RECOVER MANAGED STANDBY DATABASE . . . FINISH FORCE statement completes successfully, change the physical standby database to the primary database role by issuing the following SQL statement:

```
SQL> ALTER DATABASE COMMIT TO SWITCHOVER TO PRIMARY;
```

After issuing this SQL statement, the target standby database undergoes a transition to the primary role. As a result, you can no longer use this database as a standby database and any subsequent redo received from the original primary database cannot be applied. During the failover process, the standby redo log files were automatically archived and recovered on all other standby databases derived from the original primary database. This will happen only if the standby destinations are correctly defined on the new primary database.

There is no need to shut down and restart any of the other standby databases in the configuration that were not participants in the failover.

Step 6 Finish the transition of the standby database to the primary database role.

The task you perform in this step depends on if the physical standby database was ever opened in read-only mode:

- If the physical standby database *has not been* opened in read-only mode since the last time it was started, issue the SQL `ALTER DATABASE OPEN` statement to open the new primary database:

```
SQL> ALTER DATABASE OPEN;
```

Then, go to step 7.

- If the physical standby database *has been* opened in read-only mode since the last time it was started, you must shut down the target standby database and restart it:

```
SQL> SHUTDOWN IMMEDIATE;
```

```
SQL> STARTUP;
```

The target physical standby database has now undergone a transition to the primary database role.

See [Section 5.4.1, "Specifying Role-Based Destinations with the VALID_FOR Attribute"](#) and [Chapter 14, "LOG_ARCHIVE_DEST_n Parameter Attributes"](#) for information about using the `LOG_ARCHIVE_DEST_n VALID_FOR` attribute so the Data Guard configuration operates properly after a role transition.

Step 7 Back up the new primary database.

Before issuing the `STARTUP` statement, back up the new primary database. Performing a backup immediately is a necessary safety measure, because you cannot recover changes made after the failover without a complete backup copy of the database.

As a result of the failover, the original primary database can no longer participate in the Data Guard configuration, and all other standby databases are now receiving and applying redo data from the new primary database.

Step 8 Optionally, restore the failed primary database.

After a failover, the original primary database no longer participates in the configuration. After performing a failover, you may be able to restore the failed primary database as a new standby database using either of the following methods:

- Use Flashback Database to restore the failed primary database to a point in time before the failover occurred and then convert it into a standby database following the procedure in [Section 12.4, "Using Flashback Database After a Failover"](#) on page 12-23.

Note: You must have already enabled Flashback Database on the old primary database before the failover. See *Oracle Database Backup and Recovery Basics* for more information.

- Re-create the failed database and add it to the configuration as a new standby database. To reuse the old primary database in the new configuration, you must re-create it as a standby database using a backup copy of the new primary database. This procedure is described in [Section 3.2, "Step-by-Step Instructions for Creating a Physical Standby Database"](#) or [Section 4.2, "Step-by-Step Instructions for Creating a Logical Standby Database"](#).
- Use Oracle Enterprise Manager or the `DGMGRL REINSTATE DATABASE` command to re-create the failed primary database as a standby database in the new configuration when a connection to it is reestablished. Step-by-step instructions for reinstatement are described in *Oracle Data Guard Broker*. This option requires that Flashback Database is enabled prior to the failover.

Once the failed primary database has been restored and is running in the standby role, you can optionally perform a switchover to perform a role transition of the databases to their original (pre-failure) roles. See [Section 7.2.1, "Switchovers Involving a Physical Standby Database"](#) for more information.

7.3 Role Transitions Involving Logical Standby Databases

This section describes how to perform switchovers and failovers involving a logical standby database.

7.3.1 Switchovers Involving a Logical Standby Database

When you perform a switchover that changes roles between a primary database and a logical standby database, always initiate the switchover on the primary database and complete it on the logical standby database. These steps must be performed in the order in which they are described or the switchover will not succeed.

Note: If the primary database is a RAC database, ensure that all but one instance are shut down, and the corresponding threads are disabled before initiating the switchover. Similarly, if the logical standby database is a RAC database, ensure that all instances except the one where SQL Apply is running are shut down, and the corresponding threads are disabled before initiating the switchover. You can reenabale the threads and start the instances once the switchover operation has completed successfully. Although the instances are shut down, the role change will be automatically propagated to these instances when they are restarted.

Step 1 Verify it is possible to perform a switchover on the primary database.

On the current primary database, query the `SWITCHOVER_STATUS` column of the `V$DATABASE` fixed view on the primary database to verify it is possible to perform a switchover.

For example:

```
SQL> SELECT SWITCHOVER_STATUS FROM V$DATABASE;
SWITCHOVER_STATUS
```

```
-----  
TO STANDBY  
1 row selected
```

A value of `TO STANDBY` or `SESSIONS ACTIVE` in the `SWITCHOVER_STATUS` column indicates that it is possible to switch the primary database to the logical standby role. If one of these values is not displayed, then verify the Data Guard configuration is functioning correctly (for example, verify all `LOG_ARCHIVE_DEST_n` parameter values are specified correctly). See *Oracle Database Reference* for information about other valid values for the `SWITCHOVER_STATUS` column of the `V$DATABASE` view.

Step 2 Prepare the current primary database for the switchover.

To prepare the current primary database for a logical standby database role, issue the following SQL statement on the primary database:

```
SQL> ALTER DATABASE PREPARE TO SWITCHOVER TO LOGICAL STANDBY;
```

This statement notifies the current primary database that it will soon switch to the logical standby role and begin receiving redo data from a new primary database. You perform this step on the primary database in preparation to receive the LogMiner Multiversioned Data Dictionary to be recorded in the redo stream of the current logical standby database, as described in step 3.

The value `PREPARING SWITCHOVER` is displayed in the `V$DATABASE.SWITCHOVER_STATUS` column if this operation succeeds.

Step 3 Prepare the target logical standby database for the switchover.

Use the following statement to build a LogMiner Multiversioned Data Dictionary on the logical standby database that is the target of the switchover:

```
SQL> ALTER DATABASE PREPARE TO SWITCHOVER TO PRIMARY;
```

This statement also starts redo transport services on the logical standby database that begins transmitting its redo data to the current primary database and to other standby databases in the Data Guard configuration. The sites receiving redo data from this logical standby database accept the redo data but they do not apply it.

Depending on the work to be done and the size of the database, the switchover can take some time to complete.

The `V$DATABASE.SWITCHOVER_STATUS` on the logical standby database initially shows `PREPARING DICTIONARY` while the LogMiner Multiversioned Data Dictionary is being recorded in the redo stream. Once this has completed successfully, the `SWITCHOVER_STATUS` column shows `PREPARING SWITCHOVER`.

Step 4 Ensure the current primary database is ready for the future primary database's redo stream.

Before you can complete the role transition of the primary database to the logical standby role, verify the LogMiner Multiversioned Data Dictionary was received by the primary database by querying the `SWITCHOVER_STATUS` column of the `V$DATABASE` fixed view on the primary database. Without the receipt of the LogMiner Multiversioned Data Dictionary, the switchover cannot proceed, because the current primary database will not be able to interpret the redo records sent from the future primary database. The `SWITCHOVER_STATUS` column shows the progress of the switchover.

When the query returns the `TO LOGICAL STANDBY` value, you can proceed with Step 5. For example:

```
SQL> SELECT SWITCHOVER_STATUS FROM V$DATABASE;
SWITCHOVER_STATUS
-----
TO LOGICAL STANDBY
1 row selected
```

Note: You can cancel the switchover operation by issuing the following statements in the following order:

1. Cancel switchover on the primary database:
SQL> ALTER DATABASE PREPARE TO SWITCHOVER CANCEL;
 2. Cancel the switchover on the logical standby database:
SQL> ALTER DATABASE PREPARE TO SWITCHOVER CANCEL;
-

Step 5 Switch the primary database to the logical standby database role.

To complete the role transition of the primary database to a logical standby database, issue the following SQL statement:

```
SQL> ALTER DATABASE COMMIT TO SWITCHOVER TO LOGICAL STANDBY;
```

This statement waits for all current transactions on the primary database to end and prevents any new users from starting new transactions, and establishes a point in time where the switchover will be committed.

Executing this statement will also prevent users from making any changes to the data being maintained in the logical standby database. To ensure faster execution, ensure the primary database is in a quiet state with no update activity before issuing the switchover statement (for example, have all users temporarily log off the primary database). You can query the V\$TRANSACTIONS view for information about the status of any current in-progress transactions that could delay execution of this statement.

The primary database has now undergone a role transition to run in the standby database role.

When a primary database undergoes a role transition to a logical standby database role, you do not have to shut down and restart the database.

Step 6 Ensure all available redo has been applied to the target logical standby database that is about to become the new primary database.

After you complete the role transition of the primary database to the logical standby role and the switchover notification is received by the standby databases in the configuration, you should verify the switchover notification was processed by the target standby database by querying the SWITCHOVER_STATUS column of the V\$DATABASE fixed view on the target standby database. Once all available redo records are applied to the logical standby database, SQL Apply automatically shuts down in anticipation of the expected role transition.

The SWITCHOVER_STATUS value is updated to show progress during the switchover. When the status is TO PRIMARY, you can proceed with Step 7.

For example:

```
SQL> SELECT SWITCHOVER_STATUS FROM V$DATABASE;
SWITCHOVER_STATUS
-----
TO PRIMARY
1 row selected
```

See *Oracle Database Reference* for information about other valid values for the `SWITCHOVER_STATUS` column of the `V$DATABASE` view.

Step 7 Switch the target logical standby database to the primary database role.

On the logical standby database that you want to switch to the primary role, use the following SQL statement to switch the logical standby database to the primary role:

```
SQL> ALTER DATABASE COMMIT TO SWITCHOVER TO PRIMARY;
```

There is no need to shut down and restart any logical standby databases that are in the Data Guard configuration. Other existing logical standby databases will continue to function normally after a switchover completes. All existing physical standby databases, however, are rendered unable to participate in the Data Guard configuration after the switchover.

Step 8 Start SQL Apply on the new logical standby database.

On the new logical standby database, start SQL Apply:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY;
```

7.3.2 Failovers Involving a Logical Standby Database

This section describes how to perform failovers involving a logical standby database. A failover role transition involving a logical standby database necessitates taking corrective actions on the failed primary database and on all bystander logical standby databases. If Flashback Database was not enabled on the failed primary database, you must re-create the database from backups taken from the current primary database. Otherwise, you can follow the procedure described in [Section 12.4](#) to convert a failed primary database to be a logical standby database for the new primary database.

Depending on the protection mode for the configuration and the attributes you chose for redo transport services, it might be possible to automatically recover all or some of the primary database modifications.

If the target standby database was operating in a no data loss mode, no gaps in archived redo log files will exist and you can proceed directly to Step 2. Otherwise, begin with Step 1 to determine if any manual gap resolution steps must be performed.

Step 1 Copy and register any missing archived redo log files to the target logical standby database slated to become the new primary database.

Depending on the condition of the components in the configuration, you might have access to the archived redo log files on the primary database. If so, do the following:

1. Determine if any archived redo log files are missing on the logical standby database.
2. Copy missing log files from the primary database to the logical standby database.
3. Register the copied log files.

You can register an archived redo log files with the logical standby database by issuing the following statement. For example:

```
SQL> ALTER DATABASE REGISTER LOGICAL LOGFILE  
2> '/disk1/oracle/dbs/log-%r_%s_%t.arc';  
Database altered.
```

Step 2 Ensure all available archived redo log files were applied.

On the logical standby database you are transitioning to the primary role, verify all available archived redo log files were applied by querying the V\$LOGSTDBY_PROGRESS view. For example:

```
SQL> SELECT APPLIED_SCN, LATEST_SCN FROM V$LOGSTDBY_PROGRESS;
```

```
APPLIED_SCN LATEST_SCN
-----
190725      190725
```

When the APPLIED_SCN and LATEST_SCN values are equal, all attainable data is applied and the logical standby database now contains as much data as possible from the primary database.

Note: If SQL Apply is not active on the target logical standby database, issue the following statement on the target standby database to start SQL Apply:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY FINISH;
Database altered.
```

See [Chapter 9, "Managing a Logical Standby Database"](#) and [Chapter 12, "Data Guard Scenarios"](#) for information about the V\$LOGSTDBY_PROGRESS view.

Step 3 Enable remote destinations.

If you have not previously configured role-based destinations as described in [Section 5.4.1, "Specifying Role-Based Destinations with the VALID_FOR Attribute"](#) on page 5-14, identify the initialization parameters that correspond to the remote logical standby destinations for the new primary database, and manually enable archiving of redo data for each of these destinations.

For example, to enable archiving for the remote destination defined by the LOG_ARCHIVE_DEST_2 parameter, issue the following statement:

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2=ENABLE SCOPE=BOTH;
```

To ensure this change will persist if the new primary database is later restarted, update the appropriate text initialization parameter file or server parameter file. In general, when the database operates in the primary role, you must enable archiving to remote destinations, and when the database operates in the standby role, you must disable archiving to remote destinations.

See [Section 5.4.1, "Specifying Role-Based Destinations with the VALID_FOR Attribute"](#) on page 5-14 and [Chapter 14, "LOG_ARCHIVE_DEST_n Parameter Attributes"](#) for information about using the LOG_ARCHIVE_DEST_n VALID_FOR attribute to define role-based destinations in preparation for future role transitions.

Step 4 Activate the new primary database.

Issue the following statement on the target logical standby database (that you are transitioning to the new primary role):

```
SQL> ALTER DATABASE ACTIVATE LOGICAL STANDBY DATABASE FINISH APPLY;
```

This statement stops the RFS process, applies remaining redo data in the standby redo log file before the logical standby database becomes a primary database, stops SQL Apply, and activates the database in the primary database role.

If the `FINISH APPLY` clause is not specified, then unapplied redo from the current standby redo log file will not be applied before the standby database becomes the primary database.

Step 5 Prepare to recover the other standby databases.

Depending on how much redo data you were able to apply to the new primary database, you might be able to add other existing logical standby databases back into the Data Guard configuration to serve as standby databases for the new primary database. Perform the following steps on each logical standby database to prepare to add it back into the Data Guard configuration:

1. Create a database link on each logical standby database.

Use the `ALTER SESSION DISABLE GUARD` statement to bypass the database guard and allow modifications to the tables in the logical standby database. For example, the following creates a database link to the primary database `chicago`:

```
SQL> ALTER SESSION DISABLE GUARD;
SQL> CREATE DATABASE LINK chicago
  2> CONNECT TO username IDENTIFIED BY password USING 'chicago';
SQL> ALTER SESSION ENABLE GUARD;
```

The database user account specified in the `CREATE DATABASE LINK` statement must have the `SELECT_CATALOG_ROLE` role granted to it on the primary database.

Note: You must perform the dictionary build operation *after* the primary database has been opened but *before* any DDL statements have been executed. If any DDL statements are executed before the dictionary build operation is performed, the backup will be invalidated as a source for creating a logical standby database.

See *Oracle Database Administrator's Guide* for more information about creating database links.

2. Verify the database link.

On the logical standby database, verify the database link was configured correctly by executing the following query using the database link:

```
SQL> SELECT * FROM DBA_LOGSTDBY_PARAMETERS@chicago;
```

If the query succeeds, then that confirms the database link created in Step 1 can be used during role transitions.

Step 6 Start SQL Apply.

Start SQL Apply on each logical standby database.

For example, the following statement starts SQL Apply on the `chicago` database:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY NEW PRIMARY chicago;
```

When this statement completes, all remaining archived redo log files will have been applied. Depending on the work to be done, this operation can take some time to complete.

If the `ORA-16109` error is returned, you must re-create the logical standby database from a backup copy of the new primary database, and then add it to the Data Guard configuration.

The following example shows a failed attempt to start SQL Apply on a logical standby database in the new configuration where `chicago` is the service name that points to the new primary database:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY NEW PRIMARY chicago;
ALTER DATABASE START LOGICAL STANDBY APPLY NEW PRIMARY chicago
*
ERROR at line 1:
ORA-16109: failed to apply log data from previous primary
```

Step 7 Back up the new primary database.

Back up the new primary database immediately after the Data Guard database failover. Immediately performing a backup is a necessary safety measure, because you cannot recover changes made after the failover without a complete backup copy of the database.

Step 8 Restore the failed primary database.

After performing a failover, you can optionally restore the failed primary database as a new standby database using one of the following methods:

- Use Flashback Database to convert the failed primary database to a point in time before the failover occurred and then convert it into a standby database following the procedure in [Section 12.4, "Using Flashback Database After a Failover"](#) on page 12-23.

Note: You must have already enabled Flashback Database on the old primary database before the failover. See *Oracle Database Backup and Recovery Basics* for more information.

- Use the `DBMS_LOGSTDBY.REBUILD PL/SQL` procedure to rebuild the primary database as a new standby database. Before you run the procedure, you must verify:
 - Query the `V$STANDBY_LOG` or `V$LOGFILE` view to verify that standby redo log files have been archived
 - Query the `DBA_LOGSTDBY_EVENTS` view to verify that the LogMiner dictionary build completed successfully

See Also: The `DBMS_LOGSTDBY` package in *Oracle Database PL/SQL Packages and Types Reference* for information about the `REBUILD` subprogram

- Use Oracle Enterprise Manager or the `DGMGRL REINSTATE DATABASE` command to re-create the failed primary database as a standby database in the new configuration when a connection to it is reestablished. Step-by-step instructions for reinstatement are described in *Oracle Data Guard Broker*.

Re-create the failed database and add it to the configuration as a new standby database following the procedure in [Section 3.2, "Step-by-Step Instructions for Creating a Physical Standby Database"](#) on page 3-7 or [Section 4.2, "Step-by-Step Instructions for Creating a Logical Standby Database"](#) on page 4-3.

Once the failed primary database has been restored and is running in the standby role, you can optionally perform a switchover to transition the databases to their original (pre-failure) roles. See [Section 7.3.1, "Switchovers Involving a Logical Standby Database"](#) on page 7-13 for more information.

7.4 Using Flashback Database After a Role Transition

After a role transition, you can optionally use the `FLASHBACK DATABASE` command to revert the databases to a point in time or system change number (SCN) prior to when the role transition occurred.

In a physical standby database environment, you may need to flash back the primary database and all standby databases to maintain the Data Guard configuration. If you flash back the primary database to a certain SCN or time, you must flash back all the standby databases to either the same (or earlier) SCN or time. This way, after starting Redo Apply, the physical standby databases will automatically begin applying redo data received from the primary database.

When flashing back primary or standby databases in this way, you do not have to be aware of past switchovers. Oracle can automatically flashback across past switchovers if the SCN/time is before any past switchover.

Note: Flashback Database must be enabled on the databases before the role transition occurs. See *Oracle Database Backup and Recovery Basics* for more information.

7.4.1 Using Flashback Database After a Switchover

After a switchover, you can return databases to a time or system change number (SCN) prior to when the switchover occurred using the `FLASHBACK DATABASE` command.

If the switchover involved a physical standby database, the primary and standby database roles are preserved during the flashback operation. That is, the role in which the database is running does not change when the database is flashed back to the target SCN or time to which you flashed back the database. A database running in the physical standby role after the switchover but prior to the flashback will still be running in the physical standby database role after the Flashback Database operation.

If the switchover involved a logical standby database, flashing back changes the role of the standby database to what it was at the target SCN or time to which you flashed back the database.

7.4.2 Using Flashback Database After a Failover

You can use Flashback Database to convert the failed primary database to a point in time before the failover occurred and then convert it into a standby database. See [Section 12.4, "Using Flashback Database After a Failover"](#) for the complete step-by-step procedure.

Managing a Physical Standby Database

This chapter describes how to manage physical standby databases. This chapter contains the following topics:

- [Starting Up and Shutting Down a Physical Standby Database](#)
- [Opening a Standby Database for Read-Only or Read/Write Access](#)
- [Managing Primary Database Events That Affect the Standby Database](#)
- [Recovering Through the OPEN RESETLOGS Statement](#)
- [Monitoring the Primary and Standby Databases](#)
- [Tuning the Log Apply Rate for a Physical Standby Database](#)

The topics in this chapter describe how to use SQL statements, initialization parameters, and views to manage physical standby databases.

See *Oracle Data Guard Broker* to use the Data Guard broker to automate the management tasks described in this chapter.

8.1 Starting Up and Shutting Down a Physical Standby Database

This section describes the SQL*Plus statements used to start up and shut down a physical standby database.

8.1.1 Starting Up a Physical Standby Database

To start a physical standby database, use SQL*Plus to connect to the database with administrator privileges, and then use either the SQL*Plus `STARTUP` or `STARTUP MOUNT` statement. When used on a physical standby database:

- The `STARTUP` statement starts the database, mounts the database as a physical standby database, and opens the database for read-only access.
- The `STARTUP MOUNT` statement starts and mounts the database as a physical standby database, but does not open the database.

Once mounted, the database can receive archived redo data from the primary database. You then have the option of either starting Redo Apply or real-time apply, or opening the database for read-only access.

For example:

1. Start and mount the physical standby database:

```
SQL> STARTUP MOUNT;
```

2. Start Redo Apply or real-time apply:

To start Redo Apply, issue the following statement:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE  
2> DISCONNECT FROM SESSION;
```

To start real-time apply, issue the following statement:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE  
2> USING CURRENT LOGFILE;
```

On the primary database, query the `RECOVERY_MODE` column in the `V$ARCHIVED_DEST_STATUS` view, which displays the standby database's operation as `MANAGED_RECOVERY` for Redo Apply and `MANAGED_REAL_TIME_APPLY` for real-time apply.

See [Section 6.3](#) for information about Redo Apply, [Section 6.2.1](#) for information about real-time apply, and [Section 8.2](#) for information about opening a physical standby database for read-only or read/write access.

Note: When you first start Redo Apply on a newly created physical standby database that has not yet received any redo data from the primary database, an `ORA-01112` message may be returned. This indicates that Redo Apply is unable to determine the starting sequence number for media recovery. If this occurs, you must either manually retrieve and register an archived redo log file on the standby database, or wait for the automatic archiving to occur before restarting Redo Apply.

8.1.2 Shutting Down a Physical Standby Database

To shut down a physical standby database and stop Redo Apply, use the SQL*Plus `SHUTDOWN` statement. Control is not returned to the session that initiates a database shutdown until shutdown is complete.

If the primary database is up and running, defer the destination on the primary database and perform a log switch before shutting down the standby database.

To stop Redo Apply before shutting down the database, use the following steps:

1. Issue the following query to find out if the standby database is performing Redo Apply or real-time apply. If the `MRP0` or `MRP` process exists, then the standby database is applying redo.

```
SQL> SELECT PROCESS, STATUS FROM V$MANAGED_STANDBY;
```

2. If Redo Apply is running, cancel it as shown in the following example:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE CANCEL;
```

3. Shut down the standby database.

```
SQL> SHUTDOWN IMMEDIATE;
```

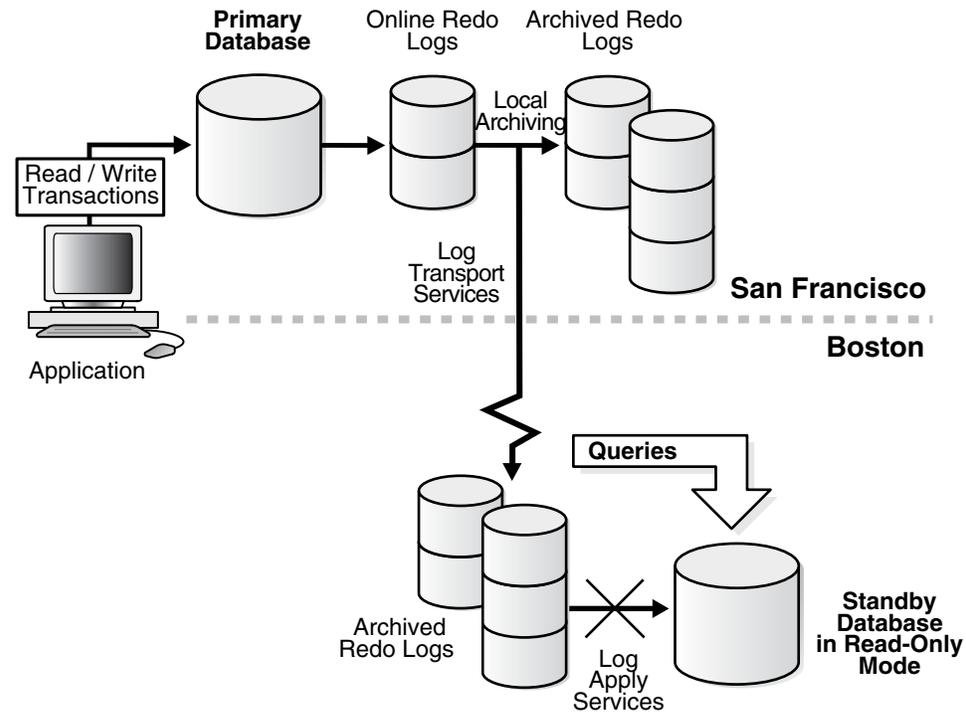
8.2 Opening a Standby Database for Read-Only or Read/Write Access

When a standby database is open for read-only access, users can query the standby database but cannot update it. Thus, you can reduce the load on the primary database by using the standby database for reporting purposes. You can periodically open the standby database for read-only access and perform ad hoc queries to verify Redo

Apply is updating the standby database correctly. (Note that for distributed queries, you must first issue the `ALTER DATABASE SET TRANSACTION READ ONLY` statement before you can issue a query on the read-only database.)

Figure 8–1 shows a standby database open for read-only access.

Figure 8–1 Standby Database Open for Read-Only Access



See Also:

- [Assessing Whether or Not to Open a Standby Database](#)
- [Opening a Physical Standby Database for Read-Only Access](#)

A physical standby database can be opened temporarily in read/write mode for development, reporting, or testing purposes, and then flashed back to a point in the past to be reverted back to a physical standby database. When the database is flashed back, Data Guard automatically synchronizes the standby database with the primary database, without the need to re-create the physical standby database from a backup copy of the primary database.

See Also: [Section 12.6](#) for a scenario that describes activating a physical standby database as a read/write reporting database, and then resynchronizing the database with the primary database

8.2.1 Assessing Whether or Not to Open a Standby Database

As you decide whether or not to open a physical standby database for read-only or read/write access, consider the following:

- Opening the physical standby database read-only may lengthen the time it takes to recover from a failure or outage, because the database must be restarted after a failover.

As long as the physical standby database has not been opened read-only since the last time it was started, a restart is unnecessary after failover, thus increasing system availability.

- While a standby database is open for read-only or read/write access, it does not apply redo data received from the primary database, thus it is not kept transactionally consistent with the primary database.

When a physical standby database is open, redo data from the primary database is received by the standby database, but the log files are not applied. At some point, you need to resume Redo Apply on the standby database, and apply the archived redo log files to resynchronize the standby database with the primary database. Because of the additional time required to apply any accumulated archived redo log files, having a standby database open for read-only access can increase the time required to complete failovers or switchovers.

You can use a physical standby database for reporting purposes or as a clone database while also maintaining the ability to complete a failover or switchover quickly if you configure more than one standby database on the standby system.

For example, based on your business requirements, you might:

- Configure two physical standby databases with one standby database always performing Redo Apply to be as current as possible with the primary database and the other standby database open in read-only mode during business hours for reporting purposes.
- Configure a physical standby database to maintain a copy of the primary database for disaster recovery purposes and also configure a logical standby database to off-load reporting tasks that require access to the latest data from the primary database.

When configuring more than one standby database on the same system, consider using the `DEPENDENCY` attribute of the `LOG_ARCHIVE_DEST_n` initialization parameter to define one archival destination to receive redo data on behalf of all of the destinations, rather than transmitting redo data to each individual destination. See [Section 5.7.5](#) for more information.

8.2.2 Opening a Physical Standby Database for Read-Only Access

You can alternate between having a physical standby database open for read-only access and performing Redo Apply using the following procedures.

To open a standby database for read-only access when it is currently shut down:

Start, mount, and open the database for read-only access using the following statement:

```
SQL> STARTUP;
```

To open a standby database for read-only access when it is currently performing Redo Apply:

1. Cancel Redo Apply:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE CANCEL;
```

2. Open the database for read-only access:

```
SQL> ALTER DATABASE OPEN;
```

You do not need to shut down the instance to open it for read-only access.

Note: By default, the `ALTER DATABASE OPEN` statement opens physical standby databases in read-only mode. The Oracle database determines if this is a physical standby database based on information in the control file.

To change the standby database from being open for read-only access to performing Redo Apply:

1. Terminate all active user sessions on the standby database.
2. Restart Redo Apply. To start Redo Apply, issue the following statement:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE
2> DISCONNECT FROM SESSION;
```

To enable real-time apply, include the `USING CURRENT LOGFILE` clause:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE
2> USING CURRENT LOGFILE;
```

You do not need to shut down the instance to start either of these apply modes.

8.3 Managing Primary Database Events That Affect the Standby Database

To prevent possible problems, you must be aware of events on the primary database that affect a standby database and learn how to respond to them. This section describes these events and the recommended responses to these events.

In some cases, the events or changes that occur on a primary database are automatically propagated through redo data to the standby database and thus require no extra action on the standby database. In other cases, you might need to perform maintenance tasks on the standby database.

[Table 8–1](#) indicates whether or not a change made on the primary database requires additional intervention by the database administrator (DBA) to be propagated to the standby database. It also briefly describes how to respond to these events. Detailed descriptions of the responses are described in the section references provided.

The following events are automatically administered by redo transport services and Redo Apply, and therefore require no intervention by the database administrator:

- A SQL `ALTER DATABASE` statement is issued with the `ENABLE THREAD` or `DISABLE THREAD` clause.
- The status of a tablespace changes (changes to read/write or read-only, placed online or taken offline).
- A datafile is added or tablespace is created when the `STANDBY_FILE_MANAGEMENT` initialization parameter is set to `AUTO`.

Table 8–1 Actions Required on a Standby Database After Changes to a Primary Database

Reference	Change Made on Primary Database	Action Required on Standby Database
Section 8.3.1	Add a datafile or create a tablespace	If you did not set the <code>STANDBY_FILE_MANAGEMENT</code> initialization parameter to <code>AUTO</code> , you must copy the new datafile to the standby database.
Section 8.3.2	Drop or delete a tablespace or datafile	Delete datafiles from primary and standby databases after the archived redo log file containing the <code>DROP</code> or <code>DELETE</code> command was applied.
Section 8.3.3	Use transportable tablespaces	Move tablespaces between the primary and standby databases.
Section 8.3.4	Rename a datafile	Rename the datafile on the standby database.
Section 8.3.5	Add or drop redo log files	Synchronize changes on the standby database.
Section 8.3.6	Perform a DML or DDL operation using the <code>NOLOGGING</code> or <code>UNRECOVERABLE</code> clause	Send the datafile containing the unlogged changes to the standby database.
Chapter 13	Change initialization parameters	Dynamically change the standby parameters or shut down the standby database and update the initialization parameter file.

8.3.1 Adding a Datafile or Creating a Tablespace

The initialization parameter, `STANDBY_FILE_MANAGEMENT`, enables you to control whether or not adding a datafile to the primary database is automatically propagated to the standby database, as follows:

- If you set the `STANDBY_FILE_MANAGEMENT` initialization parameter in the standby database server parameter file (SPFILE) to `AUTO`, any new datafiles created on the primary database are automatically created on the standby database as well.
- If you do not specify the `STANDBY_FILE_MANAGEMENT` initialization parameter or if you set it to `MANUAL`, then you must manually copy the new datafile to the standby database when you add a datafile to the primary database.

Note that if you copy an existing datafile from another database to the primary database, then you must also copy the new datafile to the standby database and re-create the standby control file, regardless of the setting of `STANDBY_FILE_MANAGEMENT` initialization parameter.

The following sections provide examples of adding a datafile to the primary and standby databases when the `STANDBY_FILE_MANAGEMENT` initialization parameter is set to `AUTO` and `MANUAL`, respectively.

8.3.1.1 When `STANDBY_FILE_MANAGEMENT` Is Set to `AUTO`

The following example shows the steps required to add a new datafile to the primary and standby databases when the `STANDBY_FILE_MANAGEMENT` initialization parameter is set to `AUTO`.

1. Add a new tablespace to the primary database:

```
SQL> CREATE TABLESPACE new_ts DATAFILE '/disk1/oracle/oradata/payroll/t_
db2.dbf'
  2> SIZE 1m AUTOEXTEND ON MAXSIZE UNLIMITED;
```

2. Archive the current online redo log file so the redo data will be transmitted to and applied on the standby database:

```
SQL> ALTER SYSTEM ARCHIVE LOG CURRENT;
```

3. Verify the new datafile was added to the primary database:

```
SQL> SELECT NAME FROM V$DATAFILE;
NAME
```

```
-----
/disk1/oracle/oradata/payroll/t_db1.dbf
/disk1/oracle/oradata/payroll/t_db2.dbf
```

4. Verify the new datafile was added to the standby database:

```
SQL> SELECT NAME FROM V$DATAFILE;
NAME
```

```
-----
/disk1/oracle/oradata/payroll/s2t_db1.dbf
/disk1/oracle/oradata/payroll/s2t_db2.dbf
```

8.3.1.2 When STANDBY_FILE_MANAGEMENT Is Set to MANUAL

This section shows how to add a new datafile to the primary and standby database when the `STANDBY_FILE_MANAGEMENT` initialization parameter is set to `MANUAL`. You must set the `STANDBY_FILE_MANAGEMENT` initialization parameter to `MANUAL` when the standby datafiles reside on raw devices. This section also describes how to recover from errors after they have occurred.

Note: Do not use the following procedure with databases that use Oracle Managed Files. Also, if the raw device path names are not the same on the primary and standby servers, use the `DB_FILE_NAME_CONVERT` initialization parameter to convert the path names.

8.3.1.2.1 Using the STANDBY_FILE_MANAGEMENT Parameter with Raw Devices

By setting the `STANDBY_FILE_MANAGEMENT` parameter to `AUTO` whenever new datafiles are added or dropped on the primary database, corresponding changes are made in the standby database without manual intervention. This is true as long as the standby database is using a file system. If the standby database is using raw devices for datafiles, then the `STANDBY_FILE_MANAGEMENT` initialization parameter will continue to work, but manual intervention is needed. This manual intervention involves ensuring the raw devices exist before log apply services on the standby database recover the redo data that will create the new datafile.

On the primary database, create a new tablespace where the datafiles reside in a raw device. At the same time, create the same raw device on the standby database. For example:

```
SQL> CREATE TABLESPACE MTS2 DATAFILE '/dev/raw/raw100' size 1m;
Tablespace created.
```

```
SQL> ALTER SYSTEM SWITCH LOGFILE;
System altered.
```

The standby database automatically adds the datafile as the raw devices exist. The standby alert log shows the following:

```
Fri Apr 8 09:49:31 2005
Media Recovery Log /u01/MILLER/flash_recovery_area/MTS_STBY/archivelog/2005_04_
08/o1_mf_1_7_15ffgt0z_.arc
Recovery created file /dev/raw/raw100
```

```
Successfully added datafile 6 to media recovery
Datafile #6: '/dev/raw/raw100'
Media Recovery Waiting for thread 1 sequence 8 (in transit)
```

However, if the raw device was created on the primary system but not on the standby, then the MRP process will shut down due to file-creation errors. For example, issue the following statements on the primary database:

```
SQL> CREATE TABLESPACE MTS3 DATAFILE '/dev/raw/raw101' size 1m;
Tablespace created.
```

```
SQL> ALTER SYSTEM SWITCH LOGFILE;
System altered.
```

The standby system does not have the `/Dave/raw/raw101` raw device created. The standby alert log shows the following messages when recovering the archive:

```
Fri Apr 8 10:00:22 2005
Media Recovery Log /u01/MILLER/flash_recovery_area/MTS_STBY/archivelog/2005_04_
08/o1_mf_1_8_15ffjrov_.arc
File #7 added to control file as 'UNNAMED00007'.
Originally created as:
'/dev/raw/raw101'
Recovery was unable to create the file as:
'/dev/raw/raw101'
MRP0: Background Media Recovery terminated with error 1274
Fri Apr 8 10:00:22 2005
Errors in file /u01/MILLER/MTS/dump/mts_mrp0_21851.trc:
ORA-01274: cannot add datafile '/dev/raw/raw101' - file could not be created
ORA-01119: error in creating database file '/dev/raw/raw101'
ORA-27041: unable to open file
Linux Error: 13: Permission denied
Additional information: 1
Some recovered datafiles maybe left media fuzzy
Media recovery may continue but open resetlogs may fail
Fri Apr 8 10:00:22 2005
Errors in file /u01/MILLER/MTS/dump/mts_mrp0_21851.trc:
ORA-01274: cannot add datafile '/dev/raw/raw101' - file could not be created
ORA-01119: error in creating database file '/dev/raw/raw101'
ORA-27041: unable to open file
Linux Error: 13: Permission denied
Additional information: 1
Fri Apr 8 10:00:22 2005
MTS; MRP0: Background Media Recovery process shutdown
ARCH: Connecting to console port...
```

8.3.1.2.2 Recovering From Errors

To correct the problems described in [Section 8.3.1.2.1](#), perform the following steps:

1. Create the raw slice on the standby database and assign permissions to the Oracle user.
2. Query the `V$DATAFILE` view. For example:

```
SQL> SELECT NAME FROM V$DATAFILE;

NAME
-----
-
/u01/MILLER/MTS/system01.dbf
/u01/MILLER/MTS/undotbs01.dbf
```

```

/u01/MILLER/MTS/sysaux01.dbf
/u01/MILLER/MTS/users01.dbf
/u01/MILLER/MTS/mts.dbf
/dev/raw/raw100
/u01/app/oracle/product/10.1.0/dbs/UNNAMED00007

SQL> ALTER SYSTEM SET STANDBY_FILE_MANAGEMENT=MANUAL;

SQL> ALTER DATABASE CREATE DATAFILE
  2  '/u01/app/oracle/product/10.1.0/dbs/UNNAMED00007'
  3  AS
  4  '/dev/raw/raw101';
    
```

3. In the standby alert log you should see information similar to the following:

```

Fri Apr  8 10:09:30 2005
alter database create datafile
'/dev/raw/raw101' as '/dev/raw/raw101'
Fri Apr  8 10:09:30 2005
Completed: alter database create datafile
'/dev/raw/raw101' a
    
```

4. On the standby database, set `STANDBY_FILE_MANAGEMENT` to `AUTO` and restart Redo Apply:

```

SQL> ALTER SYSTEM SET STANDBY_FILE_MANAGEMENT=AUTO;
SQL> RECOVER MANAGED STANDBY DATABASE DISCONNECT;
    
```

At this point Redo Apply uses the new raw device datafile and recovery continues.

8.3.2 Dropping Tablespaces and Deleting Datafiles

When you delete one or more datafiles or drop one or more tablespaces in the primary database, you also need to delete the corresponding datafiles to the standby database. The following sections provide examples of dropping tablespaces and deleting datafiles when the `STANDBY_FILE_MANAGEMENT` initialization parameter is set to `AUTO` or `MANUAL`.

8.3.2.1 When `STANDBY_FILE_MANAGEMENT` Is Set to `AUTO` or `MANUAL`

The following procedure works whether the `STANDBY_FILE_MANAGEMENT` initialization parameter is set to either `MANUAL` or `AUTO`, as follows:

1. Drop the tablespace from the primary database:

```

SQL> DROP TABLESPACE tbs_4;
SQL> ALTER SYSTEM SWITCH LOGFILE;
    
```

2. Make sure that Redo Apply is running (so that the change is applied to the standby database). If the following query returns the MRP or MRP0 process, Redo Apply is running.

```

SQL> SELECT PROCESS, STATUS FROM V$MANAGED_STANDBY;
    
```

To verify that deleted datafiles are no longer part of the database, query the `V$DATAFILE` view.

3. Delete the corresponding datafile on the standby system after the archived redo log file was applied to the standby database. For example:

```

% rm /disk1/oracle/oradata/payroll/s2tbs_4.dbf
    
```

4. On the primary database, after ensuring the standby database applied the redo information for the dropped tablespace, you can remove the datafile for the tablespace. For example:

```
% rm /disk1/oracle/oradata/payroll/tbs_4.dbf
```

8.3.2.2 Using DROP TABLESPACE INCLUDING CONTENTS AND DATAFILES

You can issue the SQL `DROP TABLESPACE INCLUDING CONTENTS AND DATAFILES` statement on the primary database to delete the datafiles on both the primary and standby databases. To use this statement, the `STANDBY_FILE_MANAGEMENT` initialization parameter must be set to `AUTO`. For example, to drop the tablespace at the primary site:

```
SQL> DROP TABLESPACE INCLUDING CONTENTS AND DATAFILES tbs_4;  
SQL> ALTER SYSTEM SWITCH LOGFILE;
```

8.3.3 Using Transportable Tablespaces with a Physical Standby Database

You can use the Oracle transportable tablespaces feature to move a subset of an Oracle database and *plug it* in to another Oracle database, essentially moving tablespaces between the databases.

To move or copy a set of tablespaces into a primary database when a physical standby is being used, perform the following steps:

1. Generate a transportable tablespace set that consists of datafiles for the set of tablespaces being transported and an export file containing structural information for the set of tablespaces.
2. Transport the tablespace set:
 - a. Copy the datafiles and the export file to the primary database.
 - b. Copy the datafiles to the standby database.

The datafiles must be copied in a directory defined by the `DB_FILE_NAME_CONVERT` initialization parameter. If `DB_FILE_NAME_CONVERT` is *not* defined, then issue the `ALTER DATABASE RENAME FILE` statement to modify the standby control file *after* the redo data containing the transportable tablespace has been applied and has failed. The `STANDBY_FILE_MANAGEMENT` initialization parameter must be set to `AUTO`.

3. Plug in the tablespace.

Invoke the Data Pump utility to plug the set of tablespaces into the primary database. Redo data will be generated and applied at the standby site to plug the tablespace into the standby database.

For more information about transportable tablespaces, see *Oracle Database Administrator's Guide*.

8.3.4 Renaming a Datafile in the Primary Database

When you rename one or more datafiles in the primary database, the change is not propagated to the standby database. Therefore, if you want to rename the same datafiles on the standby database, you must manually make the equivalent modifications on the standby database because the modifications are not performed automatically, even if the `STANDBY_FILE_MANAGEMENT` initialization parameter is set to `AUTO`.

The following steps describe how to rename a datafile in the primary database and manually propagate the changes to the standby database.

1. To rename the datafile in the primary database, take the tablespace offline:

```
SQL> ALTER TABLESPACE tbs_4 OFFLINE;
```

2. Exit from the SQL prompt and issue an operating system command, such as the following UNIX mv command, to rename the datafile on the primary system:

```
% mv /disk1/oracle/oradata/payroll/tbs_4.dbf
/disk1/oracle/oradata/payroll/tbs_x.dbf
```

3. Rename the datafile in the primary database and bring the tablespace back online:

```
SQL> ALTER TABLESPACE tbs_4 RENAME DATAFILE
  2> '/disk1/oracle/oradata/payroll/tbs_4.dbf'
  3> TO '/disk1/oracle/oradata/payroll/tbs_x.dbf';
SQL> ALTER TABLESPACE tbs_4 ONLINE;
```

4. Connect to the standby database, query the V\$ARCHIVED_LOG view to verify all of the archived redo log files are applied, and then stop Redo Apply:

```
SQL> SELECT SEQUENCE#,APPLIED FROM V$ARCHIVED_LOG ORDER BY SEQUENCE#;
SEQUENCE# APP
----- ---
8 YES
9 YES
10 YES
11 YES
4 rows selected.
```

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE CANCEL;
```

5. Shut down the standby database:

```
SQL> SHUTDOWN;
```

6. Rename the datafile at the standby site using an operating system command, such as the UNIX mv command:

```
% mv /disk1/oracle/oradata/payroll/tbs_4.dbf /disk1/oracle/oradata/payroll/tbs_x.dbf
```

7. Start and mount the standby database:

```
SQL> STARTUP MOUNT;
```

8. Rename the datafile in the standby control file. Note that the STANDBY_FILE_MANAGEMENT initialization parameter must be set to MANUAL.

```
SQL> ALTER DATABASE RENAME FILE '/disk1/oracle/oradata/payroll/tbs_4.dbf'
  2> TO '/disk1/oracle/oradata/payroll/tbs_x.dbf';
```

9. On the standby database, restart Redo Apply:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE
  2> DISCONNECT FROM SESSION;
```

If you do not rename the corresponding datafile at the standby system, and then try to refresh the standby database control file, the standby database will attempt to use the renamed datafile, but it will not find it. Consequently, you will see error messages similar to the following in the alert log:

```
ORA-00283: recovery session canceled due to errors
ORA-01157: cannot identify/lock datafile 4 - see DBWR trace file
ORA-01110: datafile 4: '/Disk1/oracle/oradata/payroll/tbs_x.dbf'
```

8.3.5 Adding or Dropping Online Redo Log Files

Changing the size and number of the online redo log files is sometimes done to tune the database. You can add or drop online redo log file groups or members to the primary database without affecting the standby database. Similarly, you can drop log file groups or members from the primary database without affecting your standby database. However, these changes do affect the performance of the standby database after switchover.

Caution: Whenever you add an online redo log file to the primary database, you should add corresponding online and standby redo log files to the standby database.

For example, if the primary database has 10 online redo log files and the standby database has 2, and then you switch over to the standby database so that it functions as the new primary database, the new primary database is forced to archive more frequently than the original primary database.

Consequently, when you add or drop an online redo log file at the primary site, it is important that you synchronize the changes in the standby database by following these steps:

1. If Redo Apply is running, you must cancel Redo Apply before you can change the log files.
2. If the `STANDBY_FILE_MANAGEMENT` initialization parameter is set to `AUTO`, change the value to `MANUAL`.
3. Add or drop an online redo log file:
 - To add an online redo log file, use a SQL statement such as this:

```
SQL> ALTER DATABASE ADD LOGFILE '/disk1/oracle/oradata/payroll/prmy3.log'
SIZE 100M;
```
 - To drop an online redo log file, use a SQL statement such as this:

```
SQL> ALTER DATABASE DROP LOGFILE '/disk1/oracle/oradata/payroll/prmy3.log';
```
4. Repeat the statement you used in Step 3 on each standby database.
5. Restore the `STANDBY_FILE_MANAGEMENT` initialization parameter and the Redo Apply options to their original states.

8.3.6 NOLOGGING or Unrecoverable Operations

When you perform a DML or DDL operation using the `NOLOGGING` or `UNRECOVERABLE` clause, the standby database is invalidated and might require substantial DBA administrative activities to repair. You can specify the `SQL ALTER DATABASE` or `SQL ALTER TABLESPACE` statement with the `FORCELOGGING` clause to override the `NOLOGGING` setting. However, this statement will not repair an already invalidated database.

See [Section 12.10](#) for information about recovering after the `NOLOGGING` clause is used.

8.4 Recovering Through the OPEN RESETLOGS Statement

Data Guard allows recovery on a physical standby database to continue after the primary database has been opened with the `RESETLOGS` option. When an `ALTER DATABASE OPEN RESETLOGS` statement is issued on the primary database, the incarnation of the database changes, creating a new branch of redo data.

When a physical standby database receives a new branch of redo data, Redo Apply automatically takes the new branch of redo data. For physical standby databases, no manual intervention is required if the standby database did not apply redo data past the new resetlogs SCN (past the start of the new branch of redo data). The following table describes how to resynchronize the standby database with the primary database branch.

If the standby database. . .	Then. . .	Perform these steps. . .
Has not applied redo data past the new resetlogs SCN (past the start of the new branch of redo data)	Redo Apply automatically takes the new branch of redo.	No manual intervention is necessary. The MRP automatically resynchronizes the standby database with the new branch of redo data.
Has applied redo data past the new resetlogs SCN (past the start of the new branch of redo data) and Flashback Database is enabled on the standby database	The standby database is recovered <i>in the future</i> of the new branch of redo data.	<ol style="list-style-type: none"> 1. Follow the procedure in Section 12.5.1 to flash back a physical standby database. 2. Restart Redo Apply to continue application of redo data onto new reset logs branch. <p>The MRP automatically resynchronizes the standby database with the new branch.</p>
Has applied redo data past the new resetlogs SCN (past the start of the new branch of redo data) and Flashback Database is not enabled on the standby database	The primary database has diverged from the standby on the indicated primary database branch.	Re-create the physical standby database following the procedures in Chapter 3 .
Is missing intervening archived redo log files from the new branch of redo data	The MRP cannot continue until the missing log files are retrieved.	Locate and register missing archived redo log files from each branch.
Is missing archived redo log files from the end of the previous branch of redo data.	The MRP cannot continue until the missing log files are retrieved.	Locate and register missing archived redo log files from the previous branch.

See *Oracle Database Backup and Recovery Advanced User's Guide* for more information about database incarnations, recovering through an `OPEN RESETLOGS` operation, and Flashback Database.

8.5 Monitoring the Primary and Standby Databases

This section gives you a general overview on where to find information for monitoring the primary and standby databases in a Data Guard environment.

This section contains the following topics:

- [Alert Log](#)
- [Dynamic Performance Views \(Fixed Views\)](#)
- [Monitoring Recovery Progress](#)
- [Monitoring Log Apply Services on Physical Standby Databases](#)

Table 8–2 summarizes common events that occur on the primary database and pointers to the files and views where you can monitor these events on the primary and standby sites.

Table 8–2 Location Where Common Actions on the Primary Database Can Be Monitored

Primary Database Event	Primary Site Information	Standby Site Information
A SQL <code>ALTER DATABASE</code> statement is issued with the <code>ENABLE THREAD</code> or <code>DISABLE THREAD</code> clause specified	<ul style="list-style-type: none"> ▪ Alert log ▪ <code>V\$THREAD</code> view 	Alert log
Current database role, protection mode and level, switchover status, and fast-start failover information	<code>V\$DATABASE</code>	<code>V\$DATABASE</code>
Redo log changed	<ul style="list-style-type: none"> ▪ Alert log ▪ <code>V\$LOG</code> view ▪ <code>STATUS</code> column of <code>V\$LOGFILE</code> view 	Alert log
<code>CREATE CONTROLFILE</code> statement issued	Alert log	Alert log
Managed recovery performed	Alert log	Alert log
Tablespace status changes made (made read/write or read-only, placed online or offline)	<ul style="list-style-type: none"> ▪ <code>DBA_TABLESPACES</code> view ▪ Alert log 	<code>V\$RECOVER_FILE</code> view
Datafile added or tablespace created	<ul style="list-style-type: none"> ▪ <code>DBA_DATA_FILES</code> view ▪ Alert log 	<code>V\$DATAFILE</code> view Alert log
Tablespace dropped	<ul style="list-style-type: none"> ▪ <code>DBA_DATA_FILES</code> view ▪ Alert log 	<code>V\$DATAFILE</code> view Alert log
Tablespace or datafile taken offline, or datafile is deleted offline	<ul style="list-style-type: none"> ▪ <code>V\$RECOVER_FILE</code> view ▪ Alert log ▪ <code>DBA_TABLESPACES</code> 	<code>V\$RECOVER_FILE</code> view <code>DBA_TABLESPACES</code>
Rename datafile	<ul style="list-style-type: none"> ▪ <code>V\$DATAFILE</code> ▪ Alert log 	<code>V\$DATAFILE</code> view Alert log
Unlogged or unrecoverable operations	<ul style="list-style-type: none"> ▪ <code>V\$DATAFILE</code> view ▪ <code>V\$DATABASE</code> view 	Alert log
Recovery progress	<ul style="list-style-type: none"> ▪ <code>V\$ARCHIVE_DEST_STATUS</code> view ▪ Alert log 	<code>V\$ARCHIVED_LOG</code> view <code>V\$LOG_HISTORY</code> view <code>V\$MANAGED_STANDBY</code> view Alert log
Redo transport status and progress	<ul style="list-style-type: none"> ▪ <code>V\$ARCHIVE_DEST_STATUS</code> view ▪ <code>V\$ARCHIVED_LOG</code> view ▪ <code>V\$ARCHIVE_DEST</code> view ▪ Alert log 	<code>V\$ARCHIVED_LOG</code> view Alert log
Auto extend a datafile	Alert log	Alert log
Issue <code>OPEN RESETLOGS</code> or <code>CLEAR UNARCHIVED LOGFILES</code> statements	Alert log	Alert log
Change initialization parameter	Alert log	Alert log

8.5.1 Alert Log

The database alert log is a chronological record of messages and errors. In addition to providing information about the Oracle database, it also includes information about operations specific to Data Guard, including the following:

- Messages related to administrative operations such as the following SQL statements: `ALTER DATABASE RECOVER MANAGED STANDBY`, `STARTUP`, `SHUTDOWN`, `ARCHIVE LOG`, and `RECOVER`
- Errors related to administrative operations that are reported by background processes, such as `ARC0`, `MRP0`, `RFS`, `LGWR`
- The completion timestamp for administrative operations

The alert log also provides pointers to the trace or dump files generated by a specific process.

8.5.2 Dynamic Performance Views (Fixed Views)

The Oracle database contains a set of underlying views. These views are often called **dynamic performance views** because they are continuously updated while a database is open and in use, and their contents relate primarily to performance. These views are also called **fixed views** because they cannot be altered or removed by the database administrator.

These view names are prefixed with either `V$` or `GV$`, for example, `V$ARCHIVE_DEST` or `GV$ARCHIVE_DEST`.

Standard dynamic performance views (`V$` fixed views) store information about the local instance. In contrast, global dynamic performance views (`GV$` fixed views), store information about all open instances in a Real Applications Cluster (RAC). Each `V$` fixed view has a corresponding `GV$` fixed view. Selects on `GV$` fixed views use parallel query slaves to obtain information on all instances. See [Chapter 16, "Views Relevant to Oracle Data Guard"](#) and *Oracle Database Reference* for additional information.

8.5.3 Monitoring Recovery Progress

This section shows some samples of the types of views discussed in [Section 8.5.2](#) for monitoring recovery progress in a Data Guard environment. It contains the following examples:

- [Monitoring the Process Activities](#)
- [Determining the Progress of Redo Apply](#)
- [Determining the Location and Creator of the Archived Redo Log Files](#)
- [Viewing Database Incarnations Before and After OPEN RESETLOGS](#)
- [Viewing the Archived Redo Log History](#)
- [Determining Which Log Files Were Applied to the Standby Database](#)
- [Determining Which Log Files Were Not Received by the Standby Site](#)

8.5.3.1 Monitoring the Process Activities

You can obtain information about Redo Apply on a standby database by monitoring the activities performed by the following processes:

Reference Name	System Process Names
ARCH	ARC0,ARC1,ARC2,...
MRP	MRP,MRP0
RFS	ORACLE{SID}

The V\$MANAGED_STANDBY view on the standby database site shows you the activities performed by both redo transport and Redo Apply processes in a Data Guard environment. The CLIENT_P column in the output of the following query identifies the corresponding primary database process.

```
SQL> SELECT PROCESS, CLIENT_PROCESS, SEQUENCE#, STATUS FROM V$MANAGED_STANDBY;
```

```
PROCESS CLIENT_P SEQUENCE# STATUS
-----
ARCH ARCH 0 CONNECTED
ARCH ARCH 0 CONNECTED
MRP0 N/A 204 WAIT_FOR_LOG
RFS LGWR 204 WRITING
RFS N/A 0 RECEIVING
```

8.5.3.2 Determining the Progress of Redo Apply

The V\$ARCHIVE_DEST_STATUS view on either a primary or standby database site provides you information such as the online redo log files that were archived, the archived redo log files that are applied, and the log sequence numbers of each. The following query output shows the standby database is two archived redo log files behind in applying the redo data received from the primary database.

```
SQL> SELECT ARCHIVED_THREAD#, ARCHIVED_SEQ#, APPLIED_THREAD#, APPLIED_SEQ#
2> FROM V$ARCHIVE_DEST_STATUS;
```

```
ARCHIVED_THREAD# ARCHIVED_SEQ# APPLIED_THREAD# APPLIED_SEQ#
-----
1 947 1 945
```

8.5.3.3 Determining the Location and Creator of the Archived Redo Log Files

Query the V\$ARCHIVED_LOG view on the standby database to find additional information about the archived redo log. Some information you can get includes the location of the archived redo log, which process created the archived redo log, redo log sequence number of each archived redo log file, when each log file was archived, and whether or not the archived redo log file was applied. For example:

```
SQL> SELECT NAME, CREATOR, SEQUENCE#, APPLIED, COMPLETION_TIME
2> FROM V$ARCHIVED_LOG;
```

```
NAME CREATOR SEQUENCE# APP COMPLETIO
-----
H:\ORACLE\ORADATA\PAYROLL\STANDBY\ARC00198.001 ARCH 198 YES 30-MAY-02
H:\ORACLE\ORADATA\PAYROLL\STANDBY\ARC00199.001 ARCH 199 YES 30-MAY-02
H:\ORACLE\ORADATA\PAYROLL\STANDBY\ARC00200.001 ARCH 200 YES 30-MAY-02
H:\ORACLE\ORADATA\PAYROLL\STANDBY\ARC00201.001 LGWR 201 YES 30-MAY-02
H:\ORACLE\ORADATA\PAYROLL\STANDBY\ARC00202.001 ARCH 202 YES 30-MAY-02
H:\ORACLE\ORADATA\PAYROLL\STANDBY\ARC00203.001 LGWR 203 YES 30-MAY-02
```

6 rows selected.

8.5.3.4 Viewing Database Incarnations Before and After OPEN RESETLOGS

Query the V\$DATABASE_INCARNATION view on the standby database to monitor database incarnations and the RESETLOGS_ID column.

The following queries were issued on the standby database before an OPEN RESETLOGS statement was issued on the primary database:

```
SQL> SELECT INCARNATION#, RESETLOGS_ID, STATUS FROM V$DATABASE_INCARNATION ;
```

```
INCARNATION# RESETLOGS_ID STATUS
-----
1          509191005 PARENT
2          509275501 CURRENT
```

```
SQL> SELECT RESETLOGS_ID, THREAD#, SEQUENCE#, STATUS, ARCHIVED FROM V$ARCHIVED_LOG
2 ORDER BY RESETLOGS_ID, SEQUENCE# ;
```

```
RESETLOGS_ID  THREAD#  SEQUENCE#  S  ARC
-----
509275501      1          1  A  YES
509275501      1          2  A  YES
509275501      1          3  A  YES
509275501      1          4  A  YES
509275501      1          5  A  YES
```

5 rows selected.

The following queries were issued on the standby database after an OPEN RESETLOGS statement was issued on the primary database and the standby database started to receive redo data on the new branch of redo:

```
SQL> SELECT INCARNATION#, RESETLOGS_ID, STATUS FROM V$DATABASE_INCARNATION ;
```

```
INCARNATION# RESETLOGS_ID STATUS
-----
1          509191005 PARENT
2          509275501 PARENT
3          509278970 CURRENT
```

```
SQL> SELECT RESETLOGS_ID, THREAD#, SEQUENCE#, STATUS, ARCHIVED FROM V$ARCHIVED_LOG
2 ORDER BY RESETLOGS_ID, SEQUENCE# ;
```

```
RESETLOGS_ID  THREAD#  SEQUENCE#  S  ARC
-----
509275501      1          1  A  YES
509275501      1          2  A  YES
509275501      1          3  A  YES
509275501      1          4  A  YES
509275501      1          5  A  YES
509278970      1          1  A  YES
509278970      1          2  A  YES
509278970      1          3  A  YES
```

8 rows selected.

8.5.3.5 Viewing the Archived Redo Log History

The V\$LOG_HISTORY on the standby site shows you a complete history of the archived redo log, including information such as the time of the first entry, the lowest SCN in the log, the highest SCN in the log, and the sequence numbers for the archived redo log files.

```
SQL> SELECT FIRST_TIME, FIRST_CHANGE#, NEXT_CHANGE#, SEQUENCE# FROM V$LOG_HISTORY;

FIRST_TIM FIRST_CHANGE# NEXT_CHANGE# SEQUENCE#
-----
13-MAY-02      190578      214480         1
13-MAY-02      214480      234595         2
13-MAY-02      234595      254713         3
.
.
.
30-MAY-02      3418615     3418874        201
30-MAY-02      3418874     3419280        202
30-MAY-02      3419280     3421165        203
203 rows selected.
```

8.5.3.6 Determining Which Log Files Were Applied to the Standby Database

Query the V\$LOG_HISTORY view on the standby database, which records the latest log sequence number that was applied. For example, issue the following query:

```
SQL> SELECT THREAD#, MAX(SEQUENCE#) AS "LAST_APPLIED_LOG"
2> FROM V$LOG_HISTORY
3> GROUP BY THREAD#;

THREAD# LAST_APPLIED_LOG
-----
1              967
```

In this example, the archived redo log file with log sequence number 967 is the most recently applied log file.

You can also use the APPLIED column in the V\$ARCHIVED_LOG fixed view on the standby database to find out which log files were applied on the standby database. For example:

```
SQL> SELECT THREAD#, SEQUENCE#, APPLIED FROM V$ARCHIVED_LOG;

THREAD# SEQUENCE# APP
-----
1          2 YES
1          3 YES
1          4 YES
1          5 YES
1          6 YES
1          7 YES
1          8 YES
1          9 YES
1         10 YES
1         11 NO
10 rows selected.
```

8.5.3.7 Determining Which Log Files Were Not Received by the Standby Site

Each archive destination has a destination ID assigned to it. You can query the DEST_ID column in the V\$ARCHIVE_DEST fixed view to find out your destination ID. You can then use this destination ID in a query on the primary database to discover log files that were not sent to a particular standby site.

For example, assume the current local archive destination ID on your primary database is 1, and the destination ID of one of your remote standby databases is 2. To find out which log files were not received by this standby destination, issue the following query on the primary database:

```
SQL> SELECT LOCAL.THREAD#, LOCAL.SEQUENCE# FROM
```

```

2> (SELECT THREAD#, SEQUENCE# FROM V$ARCHIVED_LOG WHERE DEST_ID=1) LOCAL
3> WHERE LOCAL.SEQUENCE# NOT IN
5> (SELECT SEQUENCE# FROM V$ARCHIVED_LOG WHERE DEST_ID=2 AND
6> THREAD# = LOCAL.THREAD#);
THREAD# SEQUENCE#
-----
1       12
1       13
1       14

```

The preceding example shows the log files that were not received by standby destination 2.

8.5.4 Monitoring Log Apply Services on Physical Standby Databases

To monitor the status of log apply services on a physical standby database, query the fixed views described in this section. You can also monitor the standby database using the Oracle Enterprise Manager GUI.

This section contains the following topics:

- [Accessing the V\\$DATABASE View](#)
- [Accessing the V\\$MANAGED_STANDBY Fixed View](#)
- [Accessing the V\\$ARCHIVE_DEST_STATUS Fixed View](#)
- [Accessing the V\\$ARCHIVED_LOG Fixed View](#)
- [Accessing the V\\$LOG_HISTORY Fixed View](#)
- [Accessing the V\\$DATAGUARD_STATUS Fixed View](#)

Also, see *Oracle Database Reference* for complete reference information about views.

8.5.4.1 Accessing the V\$DATABASE View

Issue the following query to show information about the protection mode, the protection level, the role of the database, and switchover status:

```

SQL> SELECT DATABASE_ROLE, DB_UNIQUE_NAME INSTANCE, OPEN_MODE, -
        PROTECTION_MODE, PROTECTION_LEVEL, SWITCHOVER_STATUS -
        FROM V$DATABASE;

```

Issue the following query to show information about fast-start failover:

```

SQL> SELECT FS_FAILOVER_STATUS FSFO_STATUS, FS_FAILOVER_CURRENT_TARGET -
        TARGET_STANDBY, FS_FAILOVER_THRESHOLD THRESHOLD, -
        FS_FAILOVER_OBSERVER_PRESENT OBS_PRES -
        FROM V$DATABASE;

```

8.5.4.2 Accessing the V\$MANAGED_STANDBY Fixed View

Query the physical standby database to monitor Redo Apply and redo transport services activity at the standby site.

```

SQL> SELECT PROCESS, STATUS, THREAD#, SEQUENCE#, BLOCK#, BLOCKS
2> FROM V$MANAGED_STANDBY;

```

PROCESS	STATUS	THREAD#	SEQUENCE#	BLOCK#	BLOCKS
RFS	ATTACHED	1	947	72	72
MRP0	APPLYING_LOG	1	946	10	72

The previous query output shows that an RFS process completed archiving a redo log file with sequence number 947. The output also shows that Redo Apply is actively applying an archived redo log file with the sequence number 946. The recovery operation is currently recovering block number 10 of the 72-block archived redo log file.

8.5.4.3 Accessing the V\$ARCHIVE_DEST_STATUS Fixed View

To quickly determine the level of synchronization for the standby database, issue the following query on the physical standby database:

```
SQL> SELECT ARCHIVED_THREAD#, ARCHIVED_SEQ#, APPLIED_THREAD#, APPLIED_SEQ#
 2> FROM V$ARCHIVE_DEST_STATUS;
```

```
ARCHIVED_THREAD# ARCHIVED_SEQ# APPLIED_THREAD# APPLIED_SEQ#
-----
1                947                1                945
```

The previous query output shows that the standby database is two archived redo log files behind the primary database.

To determine if real-time apply is enabled, query the RECOVERY_MODE column of the V\$ARCHIVE_DEST_STATUS view. It will contain the value MANAGED REAL TIME APPLY when real-time apply is enabled, as shown in the following example:

```
SQL> SELECT RECOVERY_MODE FROM V$ARCHIVE_DEST_STATUS WHERE DEST_ID=2 ;
```

```
RECOVERY_MODE
-----
MANAGED REAL TIME APPLY
```

8.5.4.4 Accessing the V\$ARCHIVED_LOG Fixed View

The V\$ARCHIVED_LOG fixed view on the physical standby database shows all the archived redo log files received from the primary database. This view is only useful after the standby site starts receiving redo data; before that time, the view is populated by old archived redo log records generated from the primary control file.

For example, you can execute the following SQL*Plus statement:

```
SQL> SELECT REGISTRAR, CREATOR, THREAD#, SEQUENCE#, FIRST_CHANGE#,
 2> NEXT_CHANGE# FROM V$ARCHIVED_LOG;
```

```
REGISTRAR CREATOR THREAD# SEQUENCE# FIRST_CHANGE# NEXT_CHANGE#
-----
RFS       ARCH     1         945       74651       74739
RFS       ARCH     1         946       74739       74772
RFS       ARCH     1         947       74772       74774
```

The previous query output shows three archived redo log files received from the primary database.

8.5.4.5 Accessing the V\$LOG_HISTORY Fixed View

Query the V\$LOG_HISTORY fixed view on the physical standby database to show all the archived redo log files that were applied:

```
SQL> SELECT THREAD#, SEQUENCE#, FIRST_CHANGE#, NEXT_CHANGE#
 2> FROM V$LOG_HISTORY;
```

```
THREAD# SEQUENCE# FIRST_CHANGE# NEXT_CHANGE#
-----
```

1 945 74651 74739

The previous query output shows that the most recently applied archived redo log file was sequence number 945.

8.5.4.6 Accessing the V\$DATAGUARD_STATUS Fixed View

The V\$DATAGUARD_STATUS fixed view displays events that would typically be triggered by any message to the alert log or server process trace files.

The following example shows output from the V\$DATAGUARD_STATUS view on a primary database:

```
SQL> SELECT MESSAGE FROM V$DATAGUARD_STATUS;

MESSAGE
-----
ARC0: Archival started
ARC1: Archival started
Archivelog destination LOG_ARCHIVE_DEST_2 validated for no-data-loss
recovery
Creating archive destination LOG_ARCHIVE_DEST_2: 'dest2'
ARCH: Transmitting activation ID 0
LGWR: Completed archiving log 3 thread 1 sequence 11
Creating archive destination LOG_ARCHIVE_DEST_2: 'dest2'
LGWR: Transmitting activation ID 6877c1fe
LGWR: Beginning to archive log 4 thread 1 sequence 12
ARC0: Evaluating archive log 3 thread 1 sequence 11
ARC0: Archive destination LOG_ARCHIVE_DEST_2: Previously completed
ARC0: Beginning to archive log 3 thread 1 sequence 11
Creating archive destination LOG_ARCHIVE_DEST_1:
'/oracle/arch/arch_1_11.arc'
ARC0: Completed archiving log 3 thread 1 sequence 11
ARC1: Transmitting activation ID 6877c1fe
15 rows selected.
```

The following example shows the contents of the V\$DATAGUARD_STATUS view on a physical standby database:

```
SQL> SELECT MESSAGE FROM V$DATAGUARD_STATUS;

MESSAGE
-----
ARC0: Archival started
ARC1: Archival started
RFS: Successfully opened standby logfile 6: '/oracle/dbs/sor12.log'
ARC1: Evaluating archive log 6 thread 1 sequence 11
ARC1: Beginning to archive log 6 thread 1 sequence 11
Creating archive destination LOG_ARCHIVE_DEST_1:
'/oracle/arch/arch_1_11.arc'
ARC1: Completed archiving log 6 thread 1 sequence 11
RFS: Successfully opened standby logfile 5: '/oracle/dbs/sor11.log'
Attempt to start background Managed Standby Recovery process
Media Recovery Log /oracle/arch/arch_1_9.arc

10 rows selected.
```

8.6 Tuning the Log Apply Rate for a Physical Standby Database

Consider using the following methods to optimize the time it takes to apply redo to physical standby databases. Also, see the Oracle Media Recovery Best Practices white paper for more information:

<http://otn.oracle.com/deploy/availability/htdocs/maa.htm>.

Set Parallel Recovery to Twice the Number of CPUs on One Standby Host

During media recovery or Redo Apply, the redo log file is read, and data blocks that require redo application are parsed out. With parallel media recovery, these data blocks are subsequently distributed evenly to all recovery processes to be read into the buffer cache. The default is serial recovery or zero parallelism, which implies that the same recovery process reads the redo, reads the data blocks from disk, and applies the redo changes.

To implement parallel media recovery or Redo Apply, add the optional `PARALLEL` clause to the recovery command. Furthermore, set the database parameter `PARALLEL_MAX_SERVERS` to at least the degree of parallelism. The following examples show how to set recovery parallelism:

```
RECOVER STANDBY DATABASE PARALLEL #CPUs * 2;
```

You should compare several serial and parallel recovery runs to determine optimal recovery performance.

Set `DB_BLOCK_CHECKING=FALSE` for Faster Redo Apply Rates

Setting the `DB_BLOCK_CHECKING=FALSE` parameter during standby or media recovery can provide as much as a twofold increase in the apply rate. The lack of block checking during recovery must be an accepted risk. Block checking should be enabled on the primary database. The `DB_BLOCK_CHECKSUM=TRUE` (the default) should be enabled for both production and standby databases. Because the `DB_BLOCK_CHECKING` parameter is dynamic, it can be toggled without shutting down the standby database.

Set `PARALLEL_EXECUTION_MESSAGE_SIZE = 4096`

When using parallel media recovery or parallel standby recovery, increasing the `PARALLEL_EXECUTION_MESSAGE_SIZE` database parameter to 4K (4096) can improve parallel recovery by as much as 20 percent. Set this parameter on both the primary and standby databases in preparation for switchover operations. Increasing this parameter requires more memory from the shared pool by each parallel execution slave process.

The `PARALLEL_EXECUTION_MESSAGE_SIZE` parameter is also used by parallel query operations and should be tested with any parallel query operations to ensure there is sufficient memory on the system. A large number of parallel query slaves on a 32-bit installation may reach memory limits and prohibit increasing the `PARALLEL_EXECUTION_MESSAGE_SIZE` from the default 2K (2048) to 4K.

Tune Disk I/O

The biggest bottlenecks encountered during recovery are read and write I/O. To relieve the bottleneck, use native asynchronous I/O and set the database parameter `DISK_ASYNC_IO` to `TRUE` (the default). The `DISK_ASYNC_IO` parameter controls whether or not disk I/O to datafiles is asynchronous. Asynchronous I/O should significantly reduce database file parallel reads and should improve overall recovery time.

Managing a Logical Standby Database

This chapter contains the following topics:

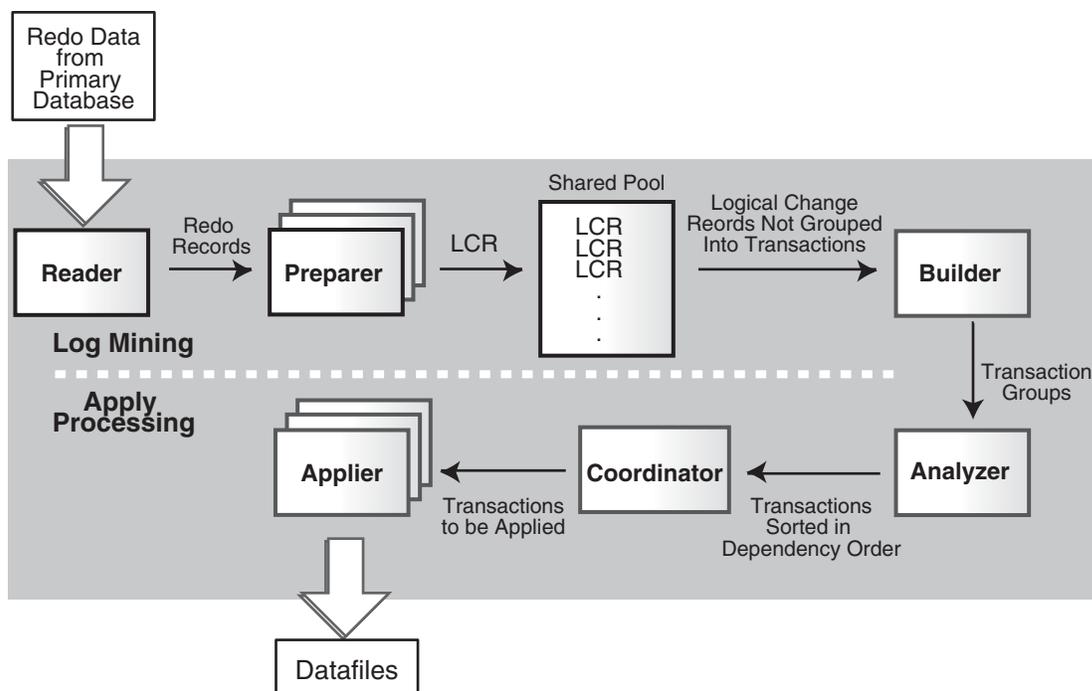
- [Overview of the SQL Apply Architecture](#)
- [Views Related to Managing and Monitoring a Logical Standby Database](#)
- [Monitoring a Logical Standby Database](#)
- [Customizing a Logical Standby Database](#)
- [Managing Specific Workloads In the Context of a Logical Standby Database](#)
- [Tuning a Logical Standby Database](#)

9.1 Overview of the SQL Apply Architecture

SQL Apply uses a collection of parallel execution servers and background processes to apply changes from the primary database to the logical standby database.

Figure 9–1 shows the flow of information and the role that each process performs.

Figure 9–1 SQL Apply Processing



The different processes involved and their functions during log mining and apply processing are as follows:

During log mining:

- The `READER` process reads redo records from the archived redo log files or standby redo log files.
- The `PREPARER` process converts block changes contained in redo records into logical change records (LCRs). Multiple `PREPARER` processes can be active for a given archived redo log file or standby redo log file. The LCRs are staged in the shared pool of the system global area (SGA), known as the *LCR cache*.
- The `BUILDER` process groups LCRs into transactions, and performs other tasks, such as memory management in the LCR cache, checkpointing related to SQL Apply restart and filtering out of uninteresting changes.

During apply processing:

- The `ANALYZER` process examines the transaction chunks containing a group of LCRs, possibly filtering out *uninteresting* transactions, and identifying dependencies between different transactions.
- The `COORDINATOR` process (LSP):
 - Assigns transactions
 - Monitors dependencies between transactions and coordinates scheduling
 - Authorizes the commitment of changes to the logical standby database
- The `APPLIER` processes:
 - Applies the LCRs to the database
 - Asks the `COORDINATOR` process to approve transactions with unresolved dependencies
 - Commits the transactions

You can query the `V$LOGSTDBY_PROCESS` view to examine the activity of the SQL Apply processes. Another view that provides information about current activity is the `V$LOGSTDBY_STATS` view that displays statistics, current state, and status information for the logical standby database during SQL Apply activities. These and other relevant views are discussed in more detail in [Section 9.2, "Views Related to Managing and Monitoring a Logical Standby Database"](#).

9.1.1 Various Considerations for SQL Apply

This section contains the following topics:

- [Transaction Size Considerations](#)
- [Pageout Considerations](#)
- [Restart Considerations](#)
- [DML Apply Considerations](#)
- [DDL Apply Considerations](#)

9.1.1.1 Transaction Size Considerations

SQL Apply categorizes transactions into two classes: small and large:

- Small transactions—SQL Apply starts applying LCRs belonging to a small transaction once it has encountered the commit record for the transaction in the redo log files.
- Large transactions—SQL Apply breaks large transactions into smaller pieces called *transaction chunks*, and starts applying the chunks before the commit record for the large transaction is seen in the redo log files. This is done to reduce memory pressure on the LCR cache and to reduce the overall failover time.

For example, without breaking into smaller pieces, a SQL*Loader load of ten million rows, each 100 bytes in size, would use more than 1 GB of memory in the LCR cache. If the memory allocated to the LCR cache was less than 1 GB, it would result in pageouts from the LCR cache.

Apart from the memory considerations, if SQL Apply did not start applying the changes related to the ten million row SQL*Loader load until it encountered the COMMIT record for the transaction, it could stall a role transition. A switchover or a failover that is initiated after the transaction commit cannot finish until SQL Apply has applied the transaction on the logical standby database.

All transactions start out categorized as small transactions. Depending on the amount of memory available for the LCR cache and the amount of memory consumed by LCRs belonging to a transaction, SQL Apply determines when to recategorize a transaction as a large transaction.

9.1.1.2 Pageout Considerations

Pageouts occur in the context of SQL Apply when memory in the LCR cache is exhausted and space needs to be released for SQL Apply to make progress.

For example, assume the memory allocated to the LCR cache is 100 MB and SQL Apply encounters an INSERT transaction to a table with a LONG column of size 300 MB. In this case, the log-mining component will page out the first part of the LONG data to read the later part of the column modification. In a well-tuned logical standby database, pageout activities will occur occasionally and should not effect the overall throughput of the system.

See Also: See [Section 9.4, "Customizing a Logical Standby Database"](#) for more information about how to identify problematic pageouts and perform corrective actions

9.1.1.3 Restart Considerations

Modifications made to the logical standby database do not become persistent until the commit record of the transaction is mined from the redo log files and applied to the logical standby database. Thus, every time SQL Apply is stopped, whether as a result of a user directive or because of a system failure, SQL Apply must go back and mine the earliest uncommitted transaction again.

In cases where a transaction does little work but remains open for a long period of time, restarting SQL Apply is prohibitively costly. This is because SQL Apply may have to mine a large number of archived redo log files again, just to read the redo data for a few uncommitted transactions. To mitigate this, SQL Apply periodically checkpoints old uncommitted data. The SCN at which the checkpoint is taken is reflected in the RESTART_SCN column of V\$LOGSTDBY_PROGRESS view.

Upon restarting, SQL Apply starts mining redo records that are generated at an SCN greater than value shown by the RESTART_SCN column. Archived redo log files that are not needed for restart are automatically deleted by SQL Apply.

Certain workloads, such as large DDL transactions, parallel DML statements (PDML), and direct-path loads, will prevent the `RESTART_SCN` from advancing for the duration of the workload.

9.1.1.4 DML Apply Considerations

SQL Apply has the following characteristics when applying DML transactions that affect the throughput and latency on the logical standby database:

- Batch updates or deletes done on the primary database, where a single statement results in multiple rows being modified, are applied as individual row modifications on the logical standby database. Thus, it is imperative for each maintained table to have a unique or a primary key. See [Section 4.1.2, "Ensure Table Rows in the Primary Database Can Be Uniquely Identified"](#) for more information.
- Direct path inserts performed on the primary database are applied using a conventional `INSERT` statement on the logical standby database.
- Parallel DML (PDML) transactions are not executed in parallel on the logical standby database.

9.1.1.5 DDL Apply Considerations

SQL Apply has the following characteristics when applying DDL transactions that affect the throughput and latency on the logical standby database:

- Parallel DDL (PDDL) transactions are *not* performed in parallel on a logical standby database.
- DDL transactions are applied serially on the logical standby database. Thus, DDL transactions applied concurrently on the primary database are applied one at a time on the logical standby database.
- `CREATE TABLE AS SELECT` (CTAS) statements are executed such that the DML activities (that are part of the CTAS statement) are suppressed on the logical standby database. The rows inserted in the newly created table as part of the CTAS statement are mined from the redo log files and applied to the logical standby database using `INSERT` statements.

9.2 Views Related to Managing and Monitoring a Logical Standby Database

The following performance views monitor the behavior of SQL Apply maintaining a logical standby database. The following sections describe the key views that can be used to monitor a logical standby database:

- [DBA_LOGSTDBY_EVENTS View](#)
- [DBA_LOGSTDBY_LOG View](#)
- [V\\$LOGSTDBY_STATS View](#)
- [V\\$LOGSTDBY_PROCESS View](#)
- [V\\$LOGSTDBY_PROGRESS View](#)
- [V\\$LOGSTDBY_STATE View](#)
- [V\\$LOGSTDBY_STATS View](#)

9.2.1 DBA_LOGSTDBY_EVENTS View

The `DBA_LOGSTDBY_EVENTS` view records interesting events that occurred during the operation of SQL Apply. By default, the view records the most recent 100 events. However, you can change the number of recorded events by calling `DBMS_LOGSTDBY.APPLY_SET()` PL/SQL procedure. If SQL Apply should stop unexpectedly, the reason for the problem is also recorded in this view.

Note: Errors that cause SQL Apply to stop are recorded in the events table. These events are put into the `ALERT.LOG` file as well, with the `LOGSTDBY` keyword included in the text. When querying the view, select the columns in order by `EVENT_TIME_STAMP`, `COMMIT_SCN`, and `CURRENT_SCN`. This ordering ensures a shutdown failure appears last in the view.

The view also contains other information, such as which DDL transactions were applied and which were skipped. For example:

```
SQL> ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-YY HH24:MI:SS';
Session altered.
SQL> COLUMN STATUS FORMAT A60
SQL> SELECT EVENT_TIME, STATUS, EVENT FROM DBA_LOGSTDBY_EVENTS
       2 ORDER BY EVENT_TIMESTAMP, COMMIT_SCN;
```

```
EVENT_TIME          STATUS
-----
EVENT
-----
23-JUL-02 18:20:12 ORA-16111: log mining and apply setting up
23-JUL-02 18:25:12 ORA-16128: User initiated shut down successfully completed
23-JUL-02 18:27:12 ORA-16112: log mining and apply stopping
23-JUL-02 18:55:12 ORA-16128: User initiated shut down successfully completed
23-JUL-02 18:57:09 ORA-16111: log mining and apply setting up
23-JUL-02 20:21:47 ORA-16204: DDL successfully applied
create table hr.test_emp (empno number, ename varchar2(64))
23-JUL-02 20:22:55 ORA-16205: DDL skipped due to skip setting
create database link link_to_boston connect to system identified by change_on_inst
7 rows selected.
```

This query shows that SQL Apply was started and stopped a few times. It also shows what DDL was applied and skipped. If SQL Apply had stopped, the last record in the query would have shown the cause of the problem.

See Also: `DBA_LOGSTDBY_EVENTS` view in *Oracle Database Reference*

9.2.2 DBA_LOGSTDBY_LOG View

The `DBA_LOGSTDBY_LOG` view provides dynamic information about archived logs being processed by SQL Apply.

For example:

```
SQL> COLUMN DICT_BEGIN FORMAT A10;
SQL> SET NUMF 9999999
SQL> SELECT FILE_NAME, SEQUENCE# AS SEQ#, FIRST_CHANGE# AS FCHANGE#, -
       NEXT_CHANGE# AS NCHANGE#, TIMESTAMP, -
       DICT_BEGIN AS BEG, DICT_END AS END, -
       THREAD# AS THR# FROM DBA_LOGSTDBY_LOG -
       ORDER BY SEQUENCE#;
```

FILE_NAME	SEQ#	F_SCN	N_SCN	TIMESTAM	BEG	END	THR#	APPLIED
/oracle/dbs/hq_nyc_2.log	2	101579	101588	11:02:58	NO	NO	1	YES
/oracle/dbs/hq_nyc_3.log	3	101588	142065	11:02:02	NO	NO	1	YES
/oracle/dbs/hq_nyc_4.log	4	142065	142307	11:02:10	NO	NO	1	YES
/oracle/dbs/hq_nyc_5.log	5	142307	142739	11:02:48	YES	YES	1	YES
/oracle/dbs/hq_nyc_6.log	6	142739	143973	12:02:10	NO	NO	1	YES
/oracle/dbs/hq_nyc_7.log	7	143973	144042	01:02:11	NO	NO	1	YES
/oracle/dbs/hq_nyc_8.log	8	144042	144051	01:02:01	NO	NO	1	YES
/oracle/dbs/hq_nyc_9.log	9	144051	144054	01:02:16	NO	NO	1	YES
/oracle/dbs/hq_nyc_10.log	10	144054	144057	01:02:21	NO	NO	1	YES
/oracle/dbs/hq_nyc_11.log	11	144057	144060	01:02:26	NO	NO	1	CURRENT
/oracle/dbs/hq_nyc_12.log	12	144060	144089	01:02:30	NO	NO	1	CURRENT
/oracle/dbs/hq_nyc_13.log	13	144089	144147	01:02:41	NO	NO	1	NO

The YES entries in the BEG and END columns indicate that a LogMiner dictionary build starts at log file sequence number 5. The most recent archived redo log file is sequence number 13, and it was received at the logical standby database at 01:02:41.

The APPLIED column indicates that SQL Apply has applied all redo before SCN 144057. Since transactions can span multiple archived log files, multiple archived log files may show the value CURRENT in the APPLIED column.

See Also: DBA_LOGSTDBY_LOG view in *Oracle Database Reference*

9.2.3 V\$LOGSTDBY_STATS View

This view provides information related to the failover characteristics of the logical standby database, including:

- The time to failover (apply finish time)
- How current is the committed data in the logical standby database (lag time)
- What the potential data loss will be in the event of a disaster (potential data loss).

For example:

```
SQL> SELECT NAME, VALUE, TIME_COMPUTED FROM V$LOGSTDBY_STATS;
```

NAME	VALUE	TIME_COMPUTED
apply finish time	+00 00:00:00.1	07-APR-2005 08:29:23
lag time	+00 00:00:00.1	07-APR-2005 08:29:23
potential data loss	+00 00:00:00	07-APR-2005 08:29:23

The unit (metric) of each of the columns displayed is in day (2) to second (1) interval. The output identifies a logical standby database that is caught up within 0.1 second of the primary database, and no data loss will occur in the event of a primary failure.

See Also: V\$LOGSTDBY_STATS view in *Oracle Database Reference*

9.2.4 V\$LOGSTDBY_PROCESS View

This view provides information about the current state of the various processes involved with SQL Apply, including:

- Identifying information (sid | serial# | spid)
- SQL Apply process: COORDINATOR, READER, BUILDER, PREPARER, ANALYZER, or APPLIER (type)

- Status of the process's current activity (`status_code | status`)
- Highest redo record processed by this process (`high_scn`)

For example:

```
SQL> COLUMN LID FORMAT 9999
SQL> COLUMN SERIAL# FORMAT 9999
SQL> COLUMN SID FORMAT 9999
SQL> SELECT SID, SERIAL#, LOGSTDBY_ID AS LID, SPID, TYPE, HIGH_SCN FROM
V$LOGSTDBY_PROCESS;
```

SID	SERIAL#	LID	SPID	TYPE	HIGH_SCN
48	6	-1	11074	COORDINATOR	7178242899
56	56	0	10858	READER	7178243497
46	1	1	10860	BUILDER	7178242901
45	1	2	10862	PREPARER	7178243295
37	1	3	10864	ANALYZER	7178241034
36	1	4	10866	APPLIER	7178239467
35	3	5	10868	APPLIER	7178239463
34	7	6	10870	APPLIER	7178239461
33	1	7	10872	APPLIER	7178239472

9 rows selected.

The `HIGH_SCN` column shows that the reader process is ahead of all other processes, and the `PREPARER` and `BUILDER` process ahead of the rest.

```
SQL> COLUMN STATUS FORMAT A40
SQL> SELECT TYPE, STATUS_CODE, STATUS FROM V$LOGSTDBY_PROCESS;
```

TYPE	STATUS_CODE	STATUS
COORDINATOR	16117	ORA-16117: processing
READER	16127	ORA-16127: stalled waiting for additional transactions to be applied
BUILDER	16116	ORA-16116: no work available
PREPARER	16116	ORA-16117: processing
ANALYZER	16120	ORA-16120: dependencies being computed for transaction at SCN 0x0001.abdb440a
APPLIER	16124	ORA-16124: transaction 1 13 1427 is waiting on another transaction
APPLIER	16121	ORA-16121: applying transaction with commit SCN 0x0001.abdb4390
APPLIER	16123	ORA-16123: transaction 1 23 1231 is waiting for commit approval
APPLIER	16116	ORA-16116: no work available

The output shows a snapshot of SQL Apply running. On the mining side, the `READER` process is waiting for additional memory to become available before it can read more, the `PREPARER` process is processing redo records, and the `BUILDER` process has no work available. On the apply side, the `COORDINATOR` is assigning more transactions to `APPLIER` processes, the `ANALYZER` is computing dependencies at SCN 7178241034, one `APPLIER` has no work available, while two have outstanding dependencies that are not yet satisfied.

See Also: `V$LOGSTDBY_PROCESS` view in *Oracle Database Reference* for reference information and [Section 9.3.1, "Monitoring SQL Apply Progress"](#) for example output

9.2.5 V\$LOGSTDBY_PROGRESS View

This view provides detailed information regarding progress made by SQL Apply, including:

- SCN or time at which all transactions that have been committed on the primary database have been applied to the logical standby database (`applied_scn` | `applied_time`)
- SCN or time at which SQL Apply would begin reading redo records (`restart_scn` | `restart_time`) on restart
- SCN or time of the latest redo record received on the logical standby database (`latest_scn` | `latest_time`)
- SCN or time of the latest record processed by the BUILDER process (`mining_scn` | `mining_time`)

For example:

```
SQL> SELECT APPLIED_SCN, LATEST_SCN, MINING_SCN, RESTART_SCN FROM V$LOGSTDBY_
PROGRESS;
```

```
APPLIED_SCN  LATEST_SCN  MINING_SCN  RESTART_SCN
-----
7178240496  7178240507  7178240507  7178219805
```

According to the output:

- SQL Apply has applied all transactions committed on or before SCN of 7178240496
- The latest redo record received at the logical standby database was generated at SCN 7178240507
- The mining component has processed all redo records generate on or before SCN 7178240507
- If SQL Apply stops and restarts for any reason, it will start mining redo records generated on or after SCN 7178219805

```
SQL> ALTER SESSION SET NLS_DATE_FORMAT='yy-mm-dd hh24:mi:ss';
Session altered
```

```
SQL> SELECT APPLIED_TIME, LATEST_TIME, MINING_TIME, RESTART_TIME FROM V$LOGSTDBY_
PROGRESS;
```

```
APPLIED_TIME      LATEST_TIME      MINING_TIME      RESTART_TIME
-----
05-05-12 10:38:21 05-05-12 10:41:21 05-05-12 10:41:53 05-05-12 10:09:30
```

According to the output:

- SQL Apply has applied all transactions committed on or before the time 05-05-12 10:38:21 (`APPLIED_TIME`)
- The last redo was generated at time 05-05-12 10:41:53 at the primary database (`LATEST_TIME`)
- The mining engine has processed all redo records generated on or before 05-05-12 10:41:21 (`MINING_TIME`)
- In the event of a restart, SQL Apply will start mining redo records generated after the time 05-05-12 10:09:30

See Also: `V$DATAGUARD_PROGRESS` view in *Oracle Database Reference* for reference information and [Section 9.3.1, "Monitoring SQL Apply Progress"](#) for example output

9.2.6 V\$LOGSTDBY_STATE View

This view provides a synopsis of the current state of SQL Apply, including:

- The DBID of the primary database (`primary_dbid`).
- The LogMiner session ID allocated to SQL Apply (`session_id`).
- Whether or not SQL Apply is applying in real time (`realtime_apply`).
- Where SQL Apply is currently with regard to loading the LogMiner Multiversioned Data Dictionary (described in [Section 4.2.3.2, "Build a Dictionary in the Redo Data"](#)), receiving redo from the primary database, and applying redo data (`STATE`)

For example:

```
SQL> COLUMN REALTIME_APPLY FORMAT a15
SQL> COLUMN STATE FORMAT a16
SQL> SELECT * FROM V$LOGSTDBY_STATE;
```

PRIMARY_DBID	SESSION_ID	REALTIME_APPLY	STATE
1562626987	1	Y	APPLYING

The output shows that SQL Apply is running in the real-time apply mode and is currently applying redo data received from the primary database, the primary database's DBID is 1562626987 and the LogMiner session identifier associated the SQL Apply session is 1.

See Also: `V$LOGSTDBY_STATE` view in *Oracle Database Reference* for reference information and [Section 9.3.1, "Monitoring SQL Apply Progress"](#) for example output

9.2.7 V\$LOGSTDBY_STATS View

This view provides SQL Apply statistics.

For example:

```
SQL> COLUMN NAME FORMAT a32
SQL> COLUMN VALUE FORMAT a32
SQL> SELECT * FROM V$LOGSTDBY_STATS;
```

NAME	VALUE
number of preparers	1
number of appliers	4
maximum SGA for LCR cache	30
parallel servers in use	8
maximum events recorded	1000
preserve commit order	TRUE
record skip errors	Y
record skip DDL	Y
record applied DDL	N
record unsupported operations	N
coordinator state	APPLYING
transactions ready	132412

```

transactions applied          132118
coordinator uptime           132102
realtime logmining           Y
apply delay                   0
Log Miner session ID         1
bytes of redo processed       130142100140
txns delivered to client     131515
DML txns delivered           128
DDL txns delivered           23
CTAS txns delivered          0
Recursive txns delivered     874
Rolled back txns seen        40
LCRs delivered to client     2246414
bytes paged out              0
secs spent in pageout        0
bytes checkpointed           0
secs spent in checkpoint     0
bytes rolled back            0
secs spent in rollback       0
secs system is idle          2119

```

32 rows selected.

See Also: V\$LOGSTDBY_STATS view in *Oracle Database Reference*

9.3 Monitoring a Logical Standby Database

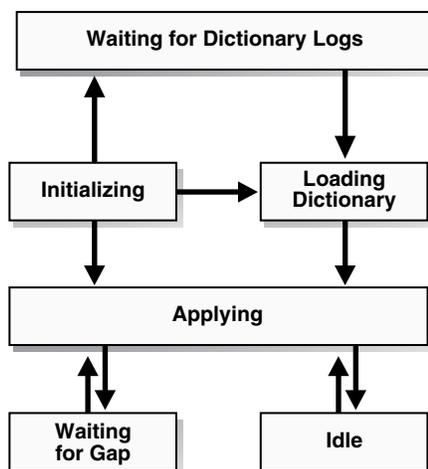
This section contains the following topics:

- [Monitoring SQL Apply Progress](#)
- [Automatic Deletion of Log Files](#)

9.3.1 Monitoring SQL Apply Progress

SQL Apply can be in any of six states of progress: initializing SQL Apply, waiting for dictionary logs, loading the LogMiner Multiversioned Data Dictionary, applying (redo data), waiting for an archive gap to be resolved, and idle. [Figure 9–2](#) shows the flow of these states.

Figure 9–2 Progress States During SQL Apply Processing



The following subsections describe each state in more detail.

Initializing State

When you start SQL Apply by issuing `ALTER DATABASE START LOGICAL STANDBY APPLY` statement, it goes in the *initializing* state.

To determine the current state of SQL Apply, query the `V$LOGSTDBY_STATE` view. For example:

```
SQL> SELECT SESSION_ID, STATE FROM V$LOGSTDBY_STATE;

SESSION_ID    STATE
-----
1             INITIALIZING
```

The `SESSION_ID` column identifies the persistent LogMiner session created by SQL Apply to mine the archived redo log files generated by the primary database.

Waiting for Dictionary Logs

The first time the SQL Apply is started, it needs to load the LogMiner MultiVersioned Data Dictionary captured in the redo log files. SQL Apply will stay in the `WAITING FOR DICTIONARY LOGS` state until it has received all redo data required to load the LogMiner MultiVersioned Data Dictionary.

Loading Dictionary State

This *loading dictionary* state can persist for a while. Loading the LogMiner multiversioned data dictionary on a large database can take a long time. Querying the `V$LOGSTDBY_STATE` view returns the following output when loading the dictionary:

```
SQL> SELECT SESSION_ID, STATE FROM V$LOGSTDBY_STATE;

SESSION_ID    STATE
-----
1             LOADING DICTIONARY
```

Only the `COORDINATOR` process and the mining processes are spawned until the LogMiner dictionary is fully loaded. Therefore, if you query the `V$LOGSTDBY_PROCESS` at this point, you will not see any of the `APPLIERS` processes. For example:

```
SQL> SELECT SID, SERIAL#, SPID, TYPE FROM V$LOGSTDBY_PROCESS;

SID    SERIAL#    SPID    TYPE
-----
47     3          11438   COORDINATOR
50     7          11334   READER
45     1          11336   BUILDER
44     2          11338   PREPARER
43     2          11340   PREPARER
```

You can get more detailed information about the progress in loading the dictionary by querying the `V$LOGMNR_DICTIONARY_LOAD` view. The dictionary load happens in three phases:

1. The relevant archived redo log files or standby redo logs files are mined to gather the redo changes relevant to load the LogMiner multiversioned data dictionary.
2. The changes are processed and loaded in staging tables inside the database.
3. The LogMiner multiversioned data dictionary tables are loaded by issuing a series of DDL statements.

For example:

```
SQL> SELECT PERCENT_DONE, COMMAND
       FROM V$LOGMNR_DICTIONARY_LOAD
       WHERE SESSION_ID = (SELECT SESSION_ID FROM V$LOGSTDBY_STATE);
```

```
PERCENT_DONE      COMMAND
-----
40                alter table SYSTEM.LOGMNR_CCOL$ exchange partition
                  P101 with table SYS.LOGMNRLT_101_CCOL$ excluding
                  indexes without validation
```

If the `PERCENT_DONE` or the `COMMAND` column does not change for a long time, query the `V$SESSION_LONGOPS` view to monitor the progress of the DDL transaction in question.

Applying State

In this state, SQL Apply has successfully loaded the initial snapshot of the LogMiner multiversioned data dictionary, and is currently applying redo data to the logical standby database.

For detailed information about the SQL Apply progress, query the `V$LOGSTDBY_PROGRESS` view:

```
SQL> ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-YYYY HH24:MI:SS';
SQL> SELECT APPLIED_TIME, APPLIED_SCN, MINING_TIME, MINING_SCN,
       FROM V$LOGSTDBY_PROGRESS;
```

```
APPLIED_TIME          APPLIED_SCN  MINING_TIME          MINING_SCN
-----
10-JAN-2005 12:00:05   346791023    10-JAN-2005 12:10:05   3468810134
```

All committed transactions seen at or before `APPLIED_SCN` (or `APPLIED_TIME`) on the primary database have been applied to the logical standby database. The mining engine has processed all redo records generated at or before `MINING_SCN` (and `MINING_TIME`) on the primary database. At steady state, the value of `MINING_SCN` (and `MINING_TIME`) will always be ahead of `APPLIED_SCN` (and `APPLIED_TIME`).

Waiting On Gap State

This state occurs when SQL Apply has mined and applied all available redo records, and is waiting for a new log file (or a missing log file) to be archived by the RFS process.

```
SQL> SELECT STATUS FROM V$LOGSTDBY_PROCESS WHERE TYPE = 'READER';
```

```
STATUS
-----
ORA:01291 Waiting for logfile
```

Idle State

SQL Apply enters this state once it has applied all redo generated by the primary database.

9.3.2 Automatic Deletion of Log Files

SQL Apply automatically deletes archived redo log files when they are no longer needed.

This behavior can be overridden by executing the following PL/SQL procedure:

```
SQL> EXECUTE DBMS_LOGSTDBY.APPLY_SET('LOG_AUTO_DELETE', FALSE);
```

Note: By default, SQL Apply will delete archived redo log files that it no longer needs. If you flash back the logical standby database, it may bring the logical standby database to a state, where an archived redo log file is present in SQL Apply metadata (as reflected in the DBA_LOGSTDBY_LOGS view) but absent in the file system. An attempt to restart SQL Apply following a Flashback Database operation may fail with the following error in the alert log:

```
Errors in file /home/oracle/DGR2/logical/stdl/bdump/stdl_lsp0_
11310.trc:
ORA-00308: cannot open archived log
'/home/oracle/DGR2/logical/stdl/stlog/1_15_559399019.dbf'
ORA-27037: unable to obtain file status
```

You need to copy the archived redo log files that have been deleted by the automatic deletion policy to the appropriate directory and restart SQL Apply.

Although SQL Apply automatically deletes archived redo log files when they are no longer needed on the logical standby database, there may be times when you want to manually remove them (for example, to reclaim disk space).

If you are overriding the default automatic log deletion capability, perform the following steps to identify and delete archived redo log files that are no longer needed by SQL Apply:

1. To purge the logical standby session of metadata that is no longer needed, enter the following PL/SQL statement:

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

This statement also updates the DBA_LOGMNR_PURGED_LOG view that displays the archived redo log files that are no longer needed.

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

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

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

3. Use an operating system-specific command to delete the archived redo log files listed by the query.

9.4 Customizing a Logical Standby Database

This section contains the following topics:

- [Using Real-Time Apply On the Logical Standby Database](#)
- [Customizing Logging of Events in the DBA_LOGSTDBY_EVENTS View](#)
- [Using DBMS_LOGSTDBY.SKIP to Prevent Changes to Specific Schema Objects](#)
- [Setting up a Skip Handler for a DDL Statement](#)
- [Modifying a Logical Standby Database](#)
- [Adding or Re-Creating Tables On a Logical Standby Database](#)

See Also: The DBMS_LOGSTDBY package in *Oracle Database PL/SQL Packages and Types Reference*

9.4.1 Using Real-Time Apply On the Logical Standby Database

By default, Data Guard waits for the *full* archived redo log file to arrive on the standby database before applying it to the standby database. However, if you have configured a standby redo log on the standby database, you can optionally enable real-time apply. With real-time apply enabled, SQL Apply applies redo data from standby redo log files at the same time the log files are being written to, as opposed to applying from an archived redo log file after a log switch occurs. Immediately applying standby redo log files in this manner keeps the logical standby database closely caught up with the primary database, without requiring the standby redo log files to be archived at the standby site. This can result in quicker switchovers and failovers.

To start real-time apply on the logical standby database, issue the following statement:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE;
```

Oracle recommends that you run SQL Apply in the real-time apply mode. See also [Section 3.1.3, "Configure a Standby Redo Log"](#) for more information about configuring a standby redo log.

9.4.2 Customizing Logging of Events in the DBA_LOGSTDBY_EVENTS View

The DBA_LOGSTDBY_EVENTS view can be thought of as a circular log containing the most recent interesting events that occurred in the context of SQL Apply. By default the last 100 events are remembered in the event view. You can change the number of events logged by invoking the DBMS_LOGSTDBY.APPLY_SET procedure. For example, to ensure that the last 10,000 events are recorded, you can issue the following statement:

```
SQL> EXECUTE DBMS_LOGSTDBY.APPLY_SET ('MAX_EVENTS_RECORDED', '10000');
```

Note: In Oracle Database 10g release 1 (10.1), the DBMS_LOGSTDBY.MAX_EVENTS constant was called DBMS_LOGSTDBY_PUBLIC.MAX_EVENTS. The effect of the two constants is the same, but in release 2 (10.2) the DBMS_LOGSTDBY_PUBLIC package has been eliminated and the definition of the constant moved to the DBMS_LOGSTDBY package.

Additionally, you can specify what types of events are recorded in the view. For example, to record applied DDL transactions to the DBA_LOGSTDBY_EVENTS view, issue the following statement:

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

Errors that cause SQL Apply to stop are always recorded in the events view (unless there is insufficient space in the system tablespace). These events are always put into the `ALERT.LOG` file as well, with the keyword `LOGSTDBY` included in the text. When querying the view, select the columns in order by `EVENT_TIME`, `COMMIT_SCN`, and `CURRENT_SCN`. This ordering ensures a shutdown failure appears last in the view.

9.4.3 Using `DBMS_LOGSTDBY.SKIP` to Prevent Changes to Specific Schema Objects

By default, all supported tables in the primary database are replicated in the logical standby database. You can change the default behavior by specifying rules to skip applying modifications to specific tables. For example, to omit changes to the `HR.EMPLOYEES` table, you can specify rules to prevent application of DML and DDL changes to the specific table. For example:

1. Stop SQL Apply:

```
SQL> ALTER DATABASE STOP LOGICAL STANDBY APPLY;
```

2. Register the `SKIP` rules:

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

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

3. Start SQL Apply:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE;
```

9.4.4 Setting up a Skip Handler for a DDL Statement

You can create a procedure to intercept certain DDL statements and replace the original DDL statement with a different one. For example, if the file system organization in the logical standby database is different than that in the primary database, you can write a `DBMS_LOGSTDBY.SKIP` procedure to transparently handle DDL transactions with file specifications.

The following procedure can handle different file system organization between the primary database and standby database, as long as you use a specific naming convention for your file-specification string.

1. Create the skip procedure to handle tablespace DDL transactions:

```
CREATE OR REPLACE PROCEDURE SYS.HANDLE_TBS_DDL (
    OLD_STMT IN VARCHAR2,
    STMT_TYP IN VARCHAR2,
    SCHEMA   IN VARCHAR2,
    NAME     IN VARCHAR2,
    XIDUSN   IN NUMBER,
    XIDSLT   IN NUMBER,
    XIDSQN   IN NUMBER,
    ACTION   OUT NUMBER,
    NEW_STMT OUT VARCHAR2
) AS
BEGIN

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

```

NEW_STMT = REPLACE(OLD_STMT,
                   '/usr/orcl/primary/dbs',
                   '/usr/orcl/stdby');

ACTION := DBMS_LOGSTDBY.SKIP_ACTION_REPLACE;

EXCEPTION
  WHEN OTHERS THEN
    ACTION := DBMS_LOGSTDBY.SKIP_ACTION_ERROR;
    NEW_STMT := NULL;
END HANDLE_TBS_DDL;

```

2. Stop SQL Apply:

```
SQL> ALTER DATABASE STOP LOGICAL STANDBY APPLY;
```

3. Register the skip procedure with SQL Apply:

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

4. Start SQL Apply:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE;
```

9.4.5 Modifying a Logical Standby Database

Logical standby database can be used for reporting activities, even while SQL statements are being applied. The *database guard* controls user access to tables in a logical standby database, and the `ALTER SESSION DATABASE DISABLE GUARD` statement is used to bypass the database guard and allow modifications to the tables in the logical standby database.

By default, a logical standby database operates with the database guard set to `ALL`, which is its most restrictive setting, and does not allow any user changes to be performed to the database. You can override the database guard to allow changes to the logical standby database by executing the `ALTER SESSION DISABLE GUARD` statement. Privileged users can issue this statement to turn the database guard off for the current session.

The following sections provide some examples. The discussions in these sections assume that the database guard is set to `ALL` or `STANDBY`.

9.4.5.1 Performing DDL on a Logical Standby Database

This section describes how to add a constraint to a table maintained through SQL Apply.

By default, only accounts with `SYS` privileges can modify the database while the database guard is set to `ALL` or `STANDBY`. If you are logged in as `SYSTEM` or another privileged account, you will not be able to issue DDL statements on the logical standby database without first bypassing the database guard for the session.

The following example shows how to stop SQL Apply, bypass the database guard, execute SQL statements on the logical standby database, and then reenables the guard:

```
SQL> ALTER DATABASE STOP LOGICAL STANDBY APPLY;
Database altered.
```

```
SQL> ALTER SESSION DISABLE GUARD;
PL/SQL procedure successfully completed.
```

```
SQL> ALTER TABLE SCOTT.EMP ADD CONSTRAINT EMPID UNIQUE (EMPNO);
Table altered.
```

```
SQL> ALTER SESSION ENABLE GUARD;
PL/SQL procedure successfully completed.
```

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY;
Database altered.
```

Oracle recommends that you do not perform DML operations on tables maintained by SQL Apply while the database guard bypass is enabled. This will introduce deviations between the primary and standby databases that will make it impossible for the logical standby database to be maintained.

9.4.5.2 Modifying Tables That Are Not Maintained by SQL Apply

Sometimes, a reporting application must collect summary results and store them temporarily or track the number of times a report was run. Although the main purpose of the application is to perform reporting activities, the application might need to issue DML (insert, update, and delete) operations on a logical standby database. It might even need to create or drop tables.

You can set up the database guard to allow reporting operations to modify data as long as the data is not being maintained through SQL Apply. To do this, you must:

- Specify the set of tables on the logical standby database to which an application can write data by executing the `DBMS_LOGSTDBY.SKIP` procedure. Skipped tables are not maintained through SQL Apply.
- Set the database guard to protect only standby tables.

In the following example, it is assumed that the tables to which the report is writing are also on the primary database.

The example stops SQL Apply, skips the tables, and then restarts SQL Apply so that changes can be applied to the logical standby database. The reporting application will be able to write to `MYTABLES%` in `MYSHEMA`. They will no longer be maintained through SQL Apply.

```
SQL> ALTER DATABASE STOP LOGICAL STANDBY APPLY;
Database altered.
```

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP(stmt => 'SCHEMA_DDL', -
    schema_name => 'HR', -
    object_name => 'TESTEMP%');
PL/SQL procedure successfully completed.
```

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP('DML', 'HR', 'TESTEMP%');
PL/SQL procedure successfully completed.
```

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE;
Database altered.
```

Once SQL Apply starts, it needs to update metadata on the standby database for the newly specified tables added in the skip rules. Attempts to modify the newly skipped table until SQL Apply has had a chance to update the metadata will fail. You can find out if SQL Apply has successfully taken into account the `SKIP` rule you just added by issuing the following query:

```
SQL> SELECT VALUE FROM DBA_LOGSDTBY_PARAMETERS
    WHERE NAME = 'GUARD_STANDBY';
```

```
VALUE
-----
Ready
```

Once the `VALUE` column displays "Ready" SQL Apply has successfully updated all relevant metadata for the skipped table, and it is safe to modify the table.

See Also: [Section C.6, "DDL Statements Supported by a Logical Standby Database"](#) and the `DBMS_LOGSTDBY` package in *Oracle Database PL/SQL Packages and Types Reference*

9.4.6 Adding or Re-Creating Tables On a Logical Standby Database

Typically, you use table instantiation to re-create a table after an unrecoverable operation. You can also use the procedure to enable SQL Apply on a table that was formerly skipped.

Before you can create a table, it must meet the requirements described in [Section 4.1.2, "Ensure Table Rows in the Primary Database Can Be Uniquely Identified"](#). Then, you can use the following steps to re-create a table named `HR.EMPLOYEES` and resume SQL Apply. The directions assume that there is already a database link `BOSTON` defined to access the primary database.

The following list shows how to re-create a table and restart SQL Apply on that table:

1. Stop SQL Apply:

```
SQL> ALTER DATABASE STOP LOGICAL STANDBY APPLY;
```

2. Ensure no operations are being skipped for the table in question by querying the `DBA_LOGSTDBY_SKIP` view:

```
SQL> SELECT * FROM DBA_LOGSTDBY_SKIP;
ERROR  STATEMENT_OPT  OWNER  NAME  PROC
-----  -
N      SCHEMA_DDL      HR      EMPLOYEES
N      DML                HR      EMPLOYEES
N      SCHEMA_DDL      OE      TEST_ORDER
N      DML                OE      TEST_ORDER
```

Because you already have skip rules associated with the table that you want to re-create on the logical standby database, you must first delete those rules. You can accomplish that by calling the `DBMS_LOGSTDBY.UNSKIP` procedure. For example:

```
SQL> EXECUTE DBMS_LOGSTDBY.UNSKIP(stmt => 'DML', -
    schema_name => 'HR', -
    object_name => 'EMPLOYEES');
SQL> EXECUTE DBMS_LOGSTDBY.UNSKIP(stmt => 'SCHEMA_DDL', -
    schema_name => 'HR', -
    object_name => 'EMPLOYEES');
```

3. Re-create the table `HR.EMPLOYEES` with all its data in the logical standby database by using the `DBMS_LOGSTDBY.INSTANTIATE_TABLE` procedure. For example:

```
SQL> EXECUTE DBMS_LOGSTDBY.INSTANTIATE_TABLE(schema_name => 'HR', -
    object_name => 'EMPLOYEES', -
    dblink => 'BOSTON');
```

4. Start SQL Apply:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE;
```

See Also: *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_LOGSTDBY.UNSKIP` and the `DBMS_LOGSTDBY.INSTANTIATE_TABLE` procedures

To ensure a consistent view across the newly instantiated table and the rest of the database, wait for SQL Apply to catch up with the primary database before querying this table. You can do this by performing the following steps:

1. On the primary database, determine the current SCN by querying the `V$DATABASE` view:

```
SQL> SELECT CURRENT_SCN FROM V$DATABASE@BOSTON;
CURRENT_SCN
-----
345162788
```

2. Make sure SQL Apply has applied all transactions committed before the `CURRENT_SCN` returned in the previous query:

```
SQL> SELECT APPLIED_SCN FROM V$LOGSTDBY_PROGRESS;

APPLIED_SCN
-----
345161345
```

When the `APPLIED_SCN` returned in this query is greater than the `CURRENT_SCN` returned in the first query, it is safe to query the newly re-created table.

9.5 Managing Specific Workloads In the Context of a Logical Standby Database

This section contains the following topics:

- [Importing a Transportable Tablespace to the Primary Database](#)
- [Using Materialized Views](#)
- [How Triggers and Constraints Are Handled on a Logical Standby Database](#)
- [Recovering Through the OPEN RESETLOGS Statement](#)

9.5.1 Importing a Transportable Tablespace to the Primary Database

Perform the following steps to import a tablespace to the primary database.

1. Disable the guard setting so that you can modify the logical standby database:

```
SQL> ALTER SESSION DISABLE GUARD;
```

2. Import the tablespace at the logical standby database.

3. Enable the database guard setting (or disconnect from the session):

```
SQL> ALTER SESSION ENABLE GUARD;
```

4. Import the tablespace at the primary database.

9.5.2 Using Materialized Views

SQL Apply does not support these DDL statements related to materialized views:

- CREATE, ALTER, or DROP MATERIALIZED VIEW
- CREATE, ALTER, or DROP MATERIALIZED VIEW LOG

Thus, new materialized views that have been created, altered, or dropped on the primary database after the logical standby database has been created are not reflected on the logical standby database. However, materialized views created on the primary database before the logical standby database has been created are also present on the logical standby database.

- For materialized views that exist on both the primary and logical standby databases, an ON-COMMIT materialized view is refreshed on the logical standby database when the transaction commit occurs.

An ON-DEMAND materialized view is not automatically refreshed by SQL Apply. You must execute the `DBMS_MVIEW.REFRESH` procedure to refresh it. For example, to refresh an ON-DEMAND materialized view named `HR.DEPARTMENTS_MV` on a logical standby database using the fast refresh method, issue the following command:

```
SQL> EXECUTE DBMS_MVIEW.REFRESH (-
        LIST => 'HR.DEPARTMENTS_MV', -
        METHOD => 'F');
```

- Additional ON-COMMIT materialized views created on the logical standby database are automatically maintained.
- Additional ON-DEMAND materialized views created on the logical standby database are not maintained by SQL Apply, and you must refresh these using the `DBMS_MVIEW.REFRESH` procedure.

9.5.3 How Triggers and Constraints Are Handled on a Logical Standby Database

By default, triggers and constraints are automatically enabled and handled on logical standby databases.

For triggers and constraints on tables *maintained* by SQL Apply:

- Constraints — Check constraints are evaluated on the primary database and do not need to be re-evaluated on the logical standby database
- Triggers — The effects of the triggers executed on the primary database are logged and applied on the standby database

For triggers and constraints on tables *not maintained* by SQL Apply:

- Constraints are evaluated
- Triggers are fired

9.5.4 Recovering Through the OPEN RESETLOGS Statement

When a logical standby database receives a new branch of redo data, SQL Apply automatically takes the new branch of redo data. For logical standby databases, no manual intervention is required if the standby database did not apply redo data past the new resetlogs SCN (past the start of the new branch of redo data). The following table describes how to resynchronize the standby database with the primary database branch.

If the standby database. . .	Then. . .	Perform these steps. . .
Has not applied redo data past the new resetlogs SCN (past the start of the new branch of redo data)	SQL Apply automatically takes the new branch of redo data.	No manual intervention is necessary. SQL Apply automatically resynchronizes the standby database with the new branch of redo data.
Has applied redo data past the new resetlogs SCN (past the start of the new branch of redo data) and Flashback Database is enabled on the standby database	The standby database is recovered <i>in the future</i> of the new branch of redo data.	<ol style="list-style-type: none"> Follow the procedure in Section 12.5.2, "Flash Back a Logical Standby Database After Flashing Back the Primary" to flash back a logical standby database. Restart SQL Apply to continue application of redo onto the new reset logs branch. <p>SQL Apply automatically resynchronizes the standby database with the new branch.</p>
Has applied redo data past the new resetlogs SCN (past the start of the new branch of redo data) and Flashback Database is not enabled on the standby database	The primary database has diverged from the standby on the indicated primary database branch.	Re-create the logical standby database following the procedures in Chapter 4, "Creating a Logical Standby Database" .
Is missing intervening archived redo log files from the new branch of redo data	SQL Apply cannot continue until the missing log files are retrieved.	Locate and register missing archived redo log files from each branch.
Is missing archived redo log files from the end of the previous branch of redo data	SQL Apply cannot continue until the missing log files are retrieved.	Locate and register missing archived redo log files from the previous branch.

See *Oracle Database Backup and Recovery Advanced User's Guide* for more information about database incarnations, recovering through an `OPEN RESETLOGS` operation, and Flashback Database.

9.6 Tuning a Logical Standby Database

This section contains the following topics:

- [Create a Primary Key RELY Constraint](#)
- [Gather Statistics for the Cost-Based Optimizer](#)
- [Adjust the Number of Processes](#)
- [Adjust the Memory Used for LCR Cache](#)
- [Adjust How Transactions are Applied On the Logical Standby Database](#)

9.6.1 Create a Primary Key RELY Constraint

On the primary database, if a table does not have a primary key or a unique index and you are certain the rows are unique, then create a primary key RELY constraint. On the logical standby database, create an index on the columns that make up the primary key. The following query generates a list of tables with no index information that can be used by a logical standby database to apply to uniquely identify rows. By creating an index on the following tables, performance can be improved significantly.

```
SQL> SELECT OWNER, TABLE_NAME FROM DBA_TABLES
2> WHERE OWNER NOT IN('SYS', 'SYSTEM', 'OUTLN', 'DBSNMP')
3> MINUS
3> SELECT DISTINCT TABLE_OWNER, TABLE_NAME FROM DBA_INDEXES
```

```

4> WHERE INDEX_TYPE NOT LIKE ('FUNCTION-BASED%')
5> MINUS
6> SELECT OWNER, TABLE_NAME FROM DBA_LOGSTDBY_UNSUPPORTED;

```

You can add a rely primary key constraint to a table on the primary database, as follows:

1. Add the primary key rely constraint at the primary database

```

SQL> ALTER TABLE HR.TEST_EMPLOYEES ADD PRIMARY KEY (EMPNO) RELY DISABLE;
SQL> ALTER SESSION DISABLE GUARD;

```

This will ensure that the EMPNO column, which can be used to uniquely identify the rows in HR.TEST_EMPLOYEES table, will be supplementally logged as part of any updates done on that table.

Note that the HR.TEST_EMPLOYEES table still does not have any unique index specified on the logical standby database. This may cause UPDATE statements to do full table scans on the logical standby database. You can remedy that by adding a unique index on the EMPNO column on the logical standby database.

See [Section 4.1.2, "Ensure Table Rows in the Primary Database Can Be Uniquely Identified"](#) and *Oracle Database SQL Reference* for more information about RELY constraints.

2. Stop SQL Apply:

```

SQL> ALTER DATABASE STOP LOGICAL STANDBY APPLY;

```

3. Disable the guard so that you can modify a maintained table on the logical standby database:

```

SQL> ALTER SESSION DISABLE GUARD;

```

4. Add a unique index on EMPNO column:

```

SQL> CREATE UNIQUE INDEX UI_TEST_EMP ON HR.TEST_EMPLOYEES (EMPNO);

```

5. Enable the guard:

```

SQL> ALTER SESSION ENABLE GUARD;

```

6. Start SQL Apply:

```

SQL> ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE;

```

9.6.2 Gather Statistics for the Cost-Based Optimizer

Statistics should be gathered on the standby database because the cost-based optimizer (CBO) uses them to determine the optimal query execution path. New statistics should be gathered after the data or structure of a schema object is modified in ways that make the previous statistics inaccurate. For example, after inserting or deleting a significant number of rows into a table, collect new statistics on the number of rows.

Statistics should be gathered on the standby database because DML and DDL operations on the primary database are executed as a function of the workload. While the standby database is logically equivalent to the primary database, SQL Apply might execute the workload in a different way. This is why using the STATS pack on the logical standby database and the V\$SYSSTAT view can be useful in determining which tables are consuming the most resources and table scans.

See Also: [Section 4.1.2, "Ensure Table Rows in the Primary Database Can Be Uniquely Identified"](#) and *Oracle Database SQL Reference* for more information about RELY constraints

9.6.3 Adjust the Number of Processes

The following sections describe:

- [Adjusting the Number of APPLIER Processes](#)
- [Adjusting the Number of PREPARER Processes](#)

9.6.3.1 Adjusting the Number of APPLIER Processes

Perform the following steps to find out whether adjusting the number of APPLIER processes will help you achieve greater throughput:

1. Determine if APPLIER processes are busy by issuing the following query:

```
SQL> SELECT COUNT(*) AS IDLE_APPLIER
      FROM V$LOGSTDBY_PROCESS
      WHERE TYPE = 'APPLIER' and status_code = 16166;
```

```
IDLE_APPLIER
-----
0
```

2. Once you are sure there are no idle APPLIER processes, issue the following query to ensure there is enough work available for additional APPLIER processes if you choose to adjust the number of APPLIERS:

```
SQL> SELECT NAME, VALUE FROM V$LOGSTDBY_STATS
      WHERE NAME LIKE 'TRANSACTIONS%';
```

```
NAME                                VALUE
-----                                -
transactions ready                   27896
transactions applied                  25671
```

These two statistics keep a cumulative total of transactions that are ready to be applied by the APPLIER processes and the number of transactions that have already been applied.

If the number (transactions ready - transactions applied) is higher than twice the number of APPLIER processes available, an improvement in throughput is possible if you increase the number of APPLIER processes.

Note: The number is a rough measure of ready work. The workload may be such that an interdependency between ready transactions will prevent additional available APPLIER processes from applying them. For instance, if the majority of the transactions that are ready to be applied are DDL transactions, adding more APPLIER processes will not result in a higher throughput.

To adjust the number of APPLIER processes to 20 from the default value of 5, perform the following steps:

- a. Stop SQL Apply:

```
SQL> ALTER DATABASE STOP LOGICAL STANDBY APPLY;
```

Database altered

- b.** Set the number of `APPLY_SERVERS` to 20:

```
SQL> EXECUTE DBMS_LOGSTDBY.APPLY_SET('APPLY_SERVERS', 20);
PL/SQL procedure successfully completed
```

- c.** Start SQL Apply:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE;
Database altered
```

9.6.3.2 Adjusting the Number of PREPARER Processes

In only rare cases do you need to adjust the number of `PREPARER` processes. Before you decide to increase the number of `PREPARER` processes, ensure the following conditions are true:

- All `PREPARER` processes are busy
- The number of transactions ready to be applied is less than the number of `APPLIERS` processes available
- There are idle `APPLIERS` processes

The following steps show how to determine these conditions are true:

- 1.** Ensure all `PREPARER` processes are busy:

```
SQL> SELECT COUNT(*) AS IDLE_PREPARER
       FROM V$LOGSTDBY_PROCESS
       WHERE TYPE = 'PREPARER' and status_code = 16166;
IDLE_PREPARER
-----
0
```

- 2.** Ensure the number of transactions ready to be applied is less than the number of `APPLIERS` processes:

```
SQL> SELECT NAME, VALUE FROM V$LOGSTDBY_STATS
       WHERE NAME LIKE 'transactions%';
NAME                                VALUE
-----                                -
transactions ready                   27896
transactions applied                 27892
```

```
SQL> SELECT COUNT(*) AS APPLIER_COUNT
       FROM V$LOGSTDBY_PROCESS WHERE TYPE = 'APPLIER';
APPLIER_COUNT
-----
20
```

Note: Issue this query several times to ensure this is not a transient event.

- 3.** Ensure there are idle `APPLIERS` processes:

```
SQL> SELECT COUNT(*) AS IDLE_APPLIER
       FROM V$LOGSTDBY_PROCESS
       WHERE TYPE = 'APPLIER' and status_code = 16166;
IDLE_APPLIER
-----
19
```

In the example, all conditions have been satisfied. Therefore, you can now increase the number of PREPARER processes to 4 (from the default value of 1), by performing the following steps:

1. Stop SQL Apply:

```
SQL> ALTER DATABASE STOP LOGICAL STANDBY APPLY;
Database altered
```

2. Set the number of PREPARE_SERVERS to 4:

```
SQL> EXECUTE DBMS_LOGSTDBY.APPLY_SET('PREPARE_SERVERS', 4);
PL/SQL procedure successfully completed
```

3. Start SQL Apply:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE;
Database altered
```

9.6.4 Adjust the Memory Used for LCR Cache

For some workloads, SQL Apply may use a large number of pageout operations, thereby reducing the overall throughput of the system. To find out whether increasing memory allocated to LCR cache will be beneficial, perform the following steps:

1. Issue the following query to obtain a snapshot of pageout activity:

```
SQL> SELECT NAME, VALUE FROM V$LOGSTDBY_STATS
        WHERE NAME LIKE '%PAGE%' OR
        NAME LIKE '%UPTIME%' OR NAME LIKE '%idle%';
```

NAME	VALUE
coordinator uptime in secs	894856
bytes paged out	20000
seconds spent in pageout	2
system idle time in secs	1000

2. Issue the query again in 5 minutes:

```
SQL> SELECT NAME, VALUE FROM V$LOGSTDBY_STATS
        WHERE NAME LIKE '%PAGE%' OR
        NAME LIKE '%UPTIME%' OR NAME LIKE '%idle%';
```

NAME	VALUE
coordinator uptime in secs	895156
bytes paged out	1020000
seconds spent in pageout	100
system idle time in secs	1000

3. Compute the normalized pageout activity. For example:

```
Change in coordinator uptime (C) = (895156 - 894856) = 300 secs
Amount of additional idle time (I) = (1000 - 1000) = 0
Change in time spent in pageout (P) = (100 - 2) = 98 secs
Pageout time in comparison to uptime = P/(C-I) = 98/300 ~ 32.67%
```

Ideally, the pageout activity should not consume more than 5 percent of the total uptime. If you continue to take snapshots over an extended interval and you find the pageout activities continue to consume a significant portion of the apply time, increasing the memory size may provide some benefits. You can increase the memory allocated to SQL Apply by performing the following steps:

1. Stop SQL Apply:

```
SQL> ALTER DATABASE STOP LOGICAL STANDBY APPLY;
Database altered
```

2. Set the memory allocated to LCR cache (for this example, the SGA is set to 1 GB):

```
SQL> EXECUTE DBMS_LOGSTDBY.APPLY_SET('MAX_SGA', 1024);
PL/SQL procedure successfully completed
```

Because the `MAX_SGA` is specified in megabytes (MB), increasing the memory to 1 GB is specified as 1024 (MB) in the example.

3. Start SQL Apply:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE;
Database altered
```

9.6.5 Adjust How Transactions are Applied On the Logical Standby Database

By default transactions are applied on the logical standby database in the exact order in which they were committed on the primary database. The default order of committing transactions allow any reporting application to run transparently on the logical standby database. However, there are times (such as after a prolonged outage of the logical standby database due to hardware failure or upgrade) when you want the logical standby database to catch up with the primary database, and can tolerate not running the reporting applications for a while. In this case, you can change the default apply mode by performing the following steps:

1. Stop SQL Apply:

```
SQL> ALTER DATABASE STOP LOGICAL STANDBY APPLY;
Database altered
```

2. Issue the following to allow transactions to be applied out of order from how they were committed on the primary databases:

```
SQL> EXECUTE DBMS_LOGSTDBY.APPLY_SET('PRESERVE_COMMIT_ORDER', 'FALSE');
PL/SQL procedure successfully completed
```

3. Start SQL Apply:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE;
Database altered
```

Once you have caught up with the primary database (verify this by querying the `V$LOGSTDBY_STATS` view), and you are ready to open the logical standby database for reporting applications, you can change the apply mode as follows:

1. Stop SQL Apply:

```
SQL> ALTER DATABASE STOP LOGICAL STANDBY APPLY;
Database altered
```

2. Restore the default value for the `PRESERVE_COMMIT_ORDER` parameter:

```
SQL> EXECUTE DBMS_LOGSTDBY.APPLY_UNSET('PRESERVE_COMMIT_ORDER');
PL/SQL procedure successfully completed
```

3. Start SQL Apply:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE;
Database altered
```

For a typical online transaction processing (OLTP) workload, the nondefault mode can provide a 50 percent or better throughput improvement over the default apply mode.

Using RMAN to Back Up and Restore Files

This chapter describes backup strategies using Oracle Recovery Manager utility (RMAN) with Data Guard and standby databases. RMAN can perform backups with minimal effect on the primary database and quickly recover from the loss of individual datafiles, or the entire database. RMAN and Data Guard can be used together to simplify the administration of a Data Guard configuration.

This chapter contains the following topics:

- [Backup Procedure](#)
- [Effect of Switchovers, Failovers, and Control File Creation on Backups](#)
- [Additional Backup Situations](#)

Note: Because a logical standby database is not a block-for-block copy of the primary database, you cannot use a logical standby database to back up the primary database.

10.1 Backup Procedure

In a standby environment, backing up datafiles and archived redo log files taken on the primary or standby system are usable on either system for recovery. Although some files such as the control file and SPFILE must be backed up on the primary database, the process of backing up datafiles and archived redo log files can be off-loaded to the standby system, to minimize the effect of backups on the production system.

Only those archived redo log files that were created by the standby instance can be backed up at the standby site. If there were any archived redo log files generated before the standby database was started, they must be backed up on the primary database. For example, if the first log sent from the primary database to the standby is log sequence 100 thread 1, then the backup of archived redo log files whose log sequence is less than 100 must be done on the primary database.

If the flash recovery area is configured, Oracle software deletes the files from the flash recovery area on an on-demand basis. The flash recovery area acts as disk cache for tape backups.

10.1.1 Using Disk as Cache for Tape Backup

The following instructions assume the flash recovery area is configured (as described in [Section 5.2.3](#)) and other RMAN persistent configurations are set. Perform the following steps:

1. On the primary database, issue the following RMAN commands to make a current backup of the control file and SPFILE, and back up files in the flash recovery area created by the primary instance to tape:

```
BACKUP DEVICE TYPE DISK CURRENT CONTROLFILE;  
BACKUP RECOVERY AREA;
```

Issue these commands (or use them in a script) every day or once a week, depending on how much application of redo data can be tolerated in the event of the loss of all current control files (see [Section 10.2.4](#)).

2. On the standby database, issue the following RMAN commands every day to roll forward a level 0 copy of the database:

```
RECOVER COPY OF DATABASE WITH TAG 'OSS';  
BACKUP DEVICE TYPE DISK INCREMENTAL LEVEL 1 FOR RECOVER OF COPY WITH TAG 'OSS'  
DATABASE;  
BACKUP DEVICE TYPE DISK ARCHIVELOG ALL NOT BACKED UP 2 TIMES;  
BACKUP RECOVERY AREA;
```

These commands apply the level 1 incremental backup taken a day before, create a new level 1 incremental backup, back up archived redo log files to the flash recovery area, and back up files created by the standby instance from the flash recovery area to tape.

10.1.2 Performing Backups Directly to Tape

If all backups are written directly to tape, configure the default device type to SBT using the RMAN `CONFIGURE DEFAULT DEVICE TYPE TO SBT` command.

On the primary database, use the following RMAN commands to back up the current control file and copy auto backups created by the primary instance to tape:

```
BACKUP AS BACKUPSET CURRENT CONTROLFILE;  
BACKUP RECOVERY AREA;
```

Issue these commands every day or once a week, depending on how much application of redo data can be tolerated in the event of loss of all current control files (refer to [Section 10.2.4](#)).

Assuming that a complete database backup is taken every Sunday, the following commands can be issued on the standby database to take a level 0 database backup:

```
BACKUP AS BACKUPSET INCREMENTAL LEVEL 0 DATABASE PLUS ARCHIVELOG NOT BACKED UP 2 TIMES;
```

On the other days of the backup cycle, run the following commands to create a level 1 incremental backup of the database and all archived redo log files that have not already been backed up 2 times:

```
BACKUP AS BACKUPSET INCREMENTAL LEVEL 1 DATABASE PLUS ARCHIVELOG NOT BACKED UP 2 TIMES;
```

10.2 Effect of Switchovers, Failovers, and Control File Creation on Backups

All the archived redo log files that were generated after the last backup on the system where backups are done must be manually cataloged using the RMAN `CATALOG ARCHIVELOG 'archivelog_name_complete_path'` command after any of the following events:

- The primary or standby control file is re-created.

- The primary database role changes to standby after a switchover.
- The standby database role changes to primary after switchover or failover.

If the new archived redo log files are not cataloged, RMAN will not back them up.

The examples in the following sections assume you are restoring files from tape to the same system on which the backup was created. If you need to restore files to a different system, you may need to either change media configuration or specify different parameters on the RMAN channels during restore, or both. See the Media Management documentation for more information about how to access RMAN backups from different systems.

10.2.1 Recovery from Loss of Datafiles on the Primary Database

Issue the following RMAN commands to restore and recover datafiles. You must be connected to both the primary and recovery catalog databases.

```
RESTORE DATAFILE n,m...;
RECOVER DATAFILE n,m...;
```

Issue the following RMAN commands to restore and recover tablespaces. You must be connected to both the primary and recovery catalog databases.

```
RESTORE TABLESPACE tbs_name1, tbs_name2, ...;
RECOVER TABLESPACE tbs_name1, tbs_name2, ...;
```

10.2.2 Recovery from Loss of Datafiles on the Standby Database

To recover the standby database after the loss of one or more datafiles, you must restore the lost files to the standby database from the backup using the RMAN `RESTORE DATAFILE` command. If all the archived redo log files required for recovery of damaged files are accessible on disk by the standby database, restart Redo Apply.

If the archived redo log files required for recovery are not accessible on disk, use RMAN to recover the restored datafiles to an SCN/log sequence greater than the last log applied to the standby database, and then restart Redo Apply to continue the application of redo data, as follows:

1. Stop Redo Apply.
2. Determine the value of the `UNTIL_SCN` column, by issuing the following query:

```
SQL> SELECT MAX(NEXT_CHANGE#)+1 UNTIL_SCN FROM V$LOG_HISTORY LH, V$DATABASE DB
WHERE LH.RESETLOGS_CHANGE#=DB.RESETLOGS_CHANGE# AND LH.RESETLOGS_TIME =
DB.RESETLOGS_TIME;
UNTIL_SCN
-----
967786
```

3. Issue the following RMAN commands to restore and recover datafiles on the standby database. You must be connected to both the standby and recovery catalog databases (use the `TARGET` keyword to connect to standby instance):

```
RESTORE DATAFILE <n,m,...>;
RECOVER DATABASE UNTIL SCN 967786;
```

To restore a tablespace, use the RMAN `'RESTORE TABLESPACE tbs_name1, tbs_name2, ...'` command.

4. Restart Redo Apply.

10.2.3 Recovery from the Loss of a Standby Control File

Oracle software allows multiplexing of the standby control file. To ensure the standby control file is multiplexed, check the `CONTROL_FILES` initialization parameter, as follows:

```
SQL> SHOW PARAMETER CONTROL_FILES
NAME                                TYPE                                VALUE
-----                                -                                -
control_files                        string                              <cfilepath1>,<cfilepath2>
```

If one of the multiplexed standby control files is lost or is not accessible, Oracle software stops the instance and writes the following messages to the alert log:

```
ORA-00210: cannot open the specified controlfile
ORA-00202: controlfile: '/ade/banand_hosted6/oracle/dbs/scf3_2.f'
ORA-27041: unable to open file
```

You can copy an intact copy of the control file over the lost copy, then restart the standby instance using the following SQL statements:

```
SQL> STARTUP MOUNT;
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE DISCONNECT FROM SESSION;
```

If all standby control files are lost, then you must create a new control file from the primary database, copy it to all multiplexed locations on the standby database, and restart the standby instance and Redo Apply. The created control file loses all information about archived redo log files generated before its creation. Because RMAN looks into the control file for the list of archived redo log files to back up, all the archived redo log files generated since the last backup must be manually cataloged.

10.2.4 Recovery from the Loss of the Primary Control File

Oracle software allows multiplexing of the control file on the primary database. If one of the control files cannot be updated on the primary database, the primary database instance is shut down automatically. As described in [Section 10.2.3](#), you can copy an intact copy of the control file and restart the instance without having to perform restore or recovery operations.

If you lose all of your control files, you can choose among the following procedures, depending on the amount of downtime that is acceptable.

Create a new control file If all control file copies are lost, you can create a new control file using the `NORESETLOGS` option and open the database after doing media recovery. An existing standby database instance can generate the script to create a new control file by using the following statement:

```
SQL> ALTER DATABASE BACKUP CONTROLFILE TO TRACE NORESETLOGS;
```

Note that if the database filenames are different in the primary and standby databases, then you must edit the generated script to correct the filenames.

This statement can be used periodically to generate a control file creation script. If you are going to use control file creation as part of your recovery plan, then you should use this statement after any physical structure change, such as adding or dropping a datafile, tablespace, or redo log member.

The created control file loses all information about the archived redo log files generated before control file creation time. If archived redo log file backups are being

performed on the primary database, all the archived redo log files generated since the last archived redo log file backup must be manually cataloged.

Recover using a backup control file If you are unable to create a control file using the previous procedure, then you can use a backup control file, perform complete recovery, and open the database with the `RESETLOGS` option.

To restore the control file and recover the database, issue the following RMAN commands after connecting to the primary instance (in `NOMOUNT` state) and catalog database:

```
RESTORE CONTROLFILE;
ALTER DATABASE MOUNT;
RECOVER DATABASE;
ALTER DATABASE OPEN RESETLOGS;
```

Beginning with Oracle Release 10.1.0, all the backups taken before a `RESETLOGS` operation can be used for recovery. Hence, it is not necessary to back up the database before making it available for production.

10.2.5 Recovery from the Loss of an Online Redo Log File

Oracle recommends multiplexing the online redo log files. The loss of all members of an online redo log group causes Oracle software to terminate the instance. If only some members of a log file group cannot be written, they will not be used until they become accessible. The views `V$LOGFILE` and `V$LOG` contain more information about the current status of log file members in the primary database instance.

When Oracle software is unable to write to one of the online redo log file members, the following alert messages are returned:

```
ORA-00313: open failed for members of log group 1 of thread 1
ORA-00312: online log 1 thread 1: '/ade/banand_hosted6/oracle/dbs/t1_log1.f'
ORA-27037: unable to obtain file status
SVR4 Error: 2: No such file or directory
Additional information: 3
```

If the access problem is temporary due to a hardware issue, correct the problem and processing will continue automatically. If the loss is permanent, a new member can be added and the old one dropped from the group.

To add a new member to a redo log group, issue the following statement:

```
SQL> ALTER DATABASE ADD LOGFILE MEMBER 'log_file_name' REUSE TO GROUP n
```

You can issue this statement even when the database is open, without affecting database availability.

If all members of an inactive group that has been archived are lost, the group can be dropped and re-created.

In all other cases (loss of all online log members for the current `ACTIVE` group, or an inactive group which has not yet been archived), you must fail over to the standby database. Refer to [Chapter 7](#) for the failover procedure.

10.2.6 Incomplete Recovery of the Database

Incomplete recovery of the primary database is normally done in cases such as when the database is logically corrupted (by some user or an application) or when a tablespace or datafile was accidentally dropped from database.

Depending on the current database checkpoint SCN on the standby database instances, you can use one of the following procedures to perform incomplete recovery of the database. All the procedures are in order of preference, starting with the one that is the least time consuming.

Using Flashback Database Using Flashback Database is the recommended procedure when the Flashback Database feature is enabled on the primary database, none of the database files are lost, and the point-in-time recovery is greater than the oldest flashback SCN or the oldest flashback time. See [Section 12.5](#) for the procedure to use Flashback Database to do point-in-time recovery.

Using the standby database instance This is the recommended procedure when the standby database is behind the desired incomplete recovery time, and Flashback Database is not enabled on the primary or standby databases:

1. Recover the standby database to the desired point in time.

```
RECOVER DATABASE UNTIL TIME 'time';
```

Alternatively, incomplete recovery time can be specified using the SCN or log sequence number:

```
RECOVER DATABASE UNTIL SCN incomplete recovery SCN
RECOVER DATABASE UNTIL LOGSEQ incomplete recovery log sequence number THREAD
thread number
```

2. Open the standby database in read-only mode to verify the state of database.

If the state is not what is desired, use the LogMiner utility to look at the archived redo log files to find the right target time or SCN for incomplete recovery. Alternatively, you can start by recovering the standby database to a point that you know is before the target time, and then open the database in read-only mode to examine the state of the data. Repeat this process until the state of the database is verified to be correct. Note that if you recover the database too far (that is, past the SCN where the error occurred) you cannot return it to an earlier SCN.

3. Activate the standby database using the SQL `ALTER DATABASE ACTIVATE STANDBY DATABASE` statement. This converts the standby database to a primary database, creates a new resetlogs branch, and opens the database. See [Section 8.4](#) to learn how the standby database reacts to the new reset logs branch.

Using the primary database instance If all of the standby database instances have already been recovered past the desired point in time and Flashback Database is enabled on the primary or standby database, then this is your only option.

Use the following procedure to perform incomplete recovery on the primary database:

1. Use LogMiner or another means to identify the time or SCN at which all the data in the database is known to be good.
2. Using the time or SCN, issue the following RMAN commands to do incomplete database recovery and open the database with the `RESETLOGS` option (after connecting to catalog database and primary instance that is in MOUNT state):

```
RUN
{
SET UNTIL TIME 'time';
RESTORE DATABASE;
RECOVER DATABASE;
}
ALTER DATABASE OPEN RESETLOGS;
```

After this process, all standby database instances must be reestablished in the Data Guard configuration.

10.3 Additional Backup Situations

The following sections describe how to modify the backup procedures for other configurations, such as when the standby and primary databases cannot share backup files; the standby instance is only used to remotely archive redo log files; or the standby database filenames are different than the primary database.

10.3.1 Standby Databases Too Geographically Distant to Share Backups

In this case, the backups taken on a standby system are not easily accessible by the primary system or other standby systems. Perform a complete backup of the database on all systems to perform recovery operations. The flash recovery area can reside locally on the primary and standby systems (for example, the flash recovery area is not the same for the primary and standby databases).

In this scenario, you can still use the general strategies described in [Section 10.2](#), with the following exceptions:

- Backup files created by RMAN must be tagged with the local system name, and with RESTORE operations that tag must be used to restrict RMAN from selecting backups taken on the same host. In other words, the BACKUP command must use the TAG *node name* option when creating backups; the RESTORE command must use the FROM TAG *node name* option; and the RECOVER command must use FROM TAG *node name* ARCHIVELOG TAG *node name* option.
- Disaster recovery of the standby site:
 1. Start the standby instance in the NOMOUNT state using the same parameter files with which the standby was operating earlier.
 2. Create a standby control file on the primary instance using the SQL ALTER DATABASE CREATE STANDBY CONTROLFILE AS *filename* statement, and use the created control file to mount the standby instance.
 3. Issue the following RMAN commands to restore and recover the database files:


```
RESTORE DATABASE FROM TAG 'node name'
RECOVER DATABASE FROM TAG 'node name' ARCHIVELOG TAG 'node name'
```
 4. Restart Redo Apply.

The standby instance will fetch the remaining archived redo log files as described in [Section 5.8](#).

10.3.2 Standby Database Does Not Contain Datafiles, Used as a FAL Server

Use the same procedure described in [Section 10.1](#), with the exception that the RMAN commands that back up database files cannot be run against the FAL server. The FAL server can be used as a backup source for all archived redo log files, thus off-loading backups of archived redo log files to the FAL server.

10.3.3 Standby Database File Names Are Different than Primary Database

If the database filenames are not the same on the primary and standby databases, the RESTORE and RECOVER commands you use will be slightly different. To obtain the actual datafile names on the standby database, query the V\$DATAFILE view and specify the SET NEWNAME option for all the datafiles in the database:

```
RUN
{
SET NEWNAME FOR DATAFILE 1 TO 'existing file location for file#1 from V$DATAFILE';
SET NEWNAME FOR DATAFILE 2 TO 'existing file location for file#2 from V$DATAFILE';
...
...
SET NEWNAME FOR DATAFILE n TO 'existing file location for file#n from V$DATAFILE';
RESTORE {DATAFILE <n,m,...> | TABLESPACE tbs_name_1, 2, ...| DATABASE;
SWITCH DATAFILE ALL;
RECOVER DATABASE {NOREDO};
}
```

Similarly, the RMAN DUPLICATE command should also use the SET NEWNAME option to specify new filenames during standby database creation.

10.3.4 Deletion Policy for Archived Redo Log Files In Flash Recovery Areas

By default, archived redo log files in a flash recovery area that were backed up to a tertiary device or made obsolete (as defined by the RMAN retention policy) are eligible for deletion. The archived redo log files that are backed up or obsolete can eventually be deleted automatically to make space if the disk space in the flash recovery area becomes full. However, you can change this *default deletion policy* using the following RMAN command:

```
CONFIGURE ARCHIVELOG DELETION POLICY TO [CLEAR | NONE | APPLIED ON STANDBY];
```

This section describes the command qualifiers and provides examples for setting up a deletion policy. See *Oracle Database Backup and Recovery Advanced User's Guide* for more information about how Oracle software manages disk space in the flash recovery area.

Using the APPLIED ON STANDBY Clause

Use the APPLIED ON STANDBY clause so that archived redo log files that have been applied on all mandatory standby destinations will be deleted. The actions taken when you specify this clause are described in the following table:

When the APPLIED ON STANDBY clause is configured on. . .	Then, these files are eligible for deletion. . .
The primary database	Archived redo log files in the flash recovery area that were applied on <i>all mandatory standby databases</i> .
A standby database that has one or more mandatory cascading standby databases	Archived redo log files in the flash recovery area that were applied on <i>all mandatory cascading standby databases</i> .
A standby database that has no cascading standby databases	Archived redo log files in the flash recovery area that were applied on <i>the standby database</i> .

See [Appendix E](#) for more information about cascaded destinations.

Using the CLEAR Clause

Use the CLEAR clause to disable the deletion policy that was previously set up with the RMAN CONFIGURE ARCHIVELOG DELETION POLICY command. The Oracle

database will resume the default deletion policy behavior, which is to delete archived redo log files that are backed up or obsolete to make space if disk space in the flash recovery area becomes full.

Using the NONE Clause

Use the NONE clause so that archived redo logs in flash recovery area that were backed up or obsolete as per the RMAN retention policy are eligible for deletion. This is the default configuration. Archived redo log files that are backed up or obsolete are deleted to make space if the disk space in the flash recovery area becomes full.

Examples of the CONFIGURE ARCHIVELOG DELETION POLICY Command

When backups of archived redo log files are taken on the standby database:

1. Issue the following command on the primary database:

```
CONFIGURE ARCHIVELOG DELETION POLICY TO APPLIED ON STANDBY;
```

2. Issue the following command on the standby database:

```
CONFIGURE ARCHIVELOG DELETION POLICY TO NONE;
```

When backups of archived redo log files are taken on the primary database:

1. Issue the following command on the standby database:

```
CONFIGURE ARCHIVELOG DELETION POLICY TO APPLIED ON STANDBY;
```

2. Issue the following command on the primary database:

```
CONFIGURE ARCHIVELOG DELETION POLICY TO NONE;
```

10.3.4.1 Reconfiguring the Deletion Policy After a Role Transition

After a switchover or failover, you may need to reissue the RMAN CONFIGURE ARCHIVELOG DELETION POLICY command on each database. If the backup site for archived redo log files remains the same, then do nothing. Otherwise, you must switch the archivelog deletion policy by issuing the CONFIGURE ARCHIVELOG DELETION POLICY TO APPLIED ON STANDBY statement on the database where backups are *not* taken, and issuing the CONFIGURE ARCHIVELOG DELETION POLICY TO NONE statement on the database where backups *are* taken.

10.3.4.2 Viewing the Current Deletion Policy

To see the current setting (APPLIED ON STANDBY, CLEAR, NONE) for a database, issue the following query:

```
SELECT NAME, VALUE FROM V$RMAN_CONFIGURATION WHERE
NAME LIKE '%ARCHIVELOG DELETION POLICY%';
```

NAME	VALUE
ARCHIVELOG DELETION POLICY	TO APPLIED ON STANDBY

You can also find the existing configuration using the RMAN SHOW ARCHIVELOG DELETION POLICY command:

```
RMAN> SHOW ARCHIVELOG DELETION POLICY
RMAN configuration parameters are:
CONFIGURE ARCHIVELOG DELETION POLICY TO APPLIED ON STANDBY;
```

Using SQL Apply to Upgrade the Oracle Database

Starting with Oracle Database 10g release 1 (10.1.0.3), you can use a logical standby database to perform a *rolling upgrade* of Oracle Database 10g software. During a rolling upgrade, you can run different releases of an Oracle database on the primary and logical standby databases while you upgrade them, one at a time, incurring minimal downtime on the primary database.

Note: This chapter describes how to upgrade with a logical standby database in place in a rolling fashion to minimize downtime. A second method, described in [Section B.3, "Upgrading Oracle Database with a Logical Standby Database In Place"](#) is a traditional upgrade procedure that results in downtime during the upgrade process. Use the steps from only one method to perform the complete upgrade. Do not attempt to use both methods or to combine the steps from the two methods.

The instructions in this chapter describe how to minimize downtime while upgrading an Oracle database. This chapter provides the following topics:

- [Benefits of a Rolling Upgrade Using SQL Apply](#)
- [Requirements to Perform a Rolling Upgrade Using SQL Apply](#)
- [Figures and Conventions Used in the Upgrade Instructions](#)
- [Prepare to Upgrade](#)
- [Upgrade the Databases](#)

11.1 Benefits of a Rolling Upgrade Using SQL Apply

Performing a rolling upgrade with SQL Apply provides several advantages:

- Your database will incur very little downtime. The overall downtime can be as little as the time it takes to perform a switchover.
- You eliminate application downtime due to PL/SQL recompilation.
- You can validate the upgraded database release without affecting the primary database.

11.2 Requirements to Perform a Rolling Upgrade Using SQL Apply

The rolling upgrade procedure requires the following:

- A primary database that is running Oracle Database release *x* and a logical standby database that is running Oracle Database release *y*.
- The databases must not be part of a Data Guard Broker configuration. See *Oracle Data Guard Broker* for information about removing databases from a broker configuration.
- The Data Guard protection mode must be set to either maximum availability or maximum performance. Query the `PROTECTION_LEVEL` column in the `V$DATABASE` view to find out the current protection mode setting.
- The `LOG_ARCHIVE_DEST_n` initialization parameter for the logical standby database destination must be set to `OPTIONAL` to ensure the primary database can proceed while the logical standby database is being upgraded.

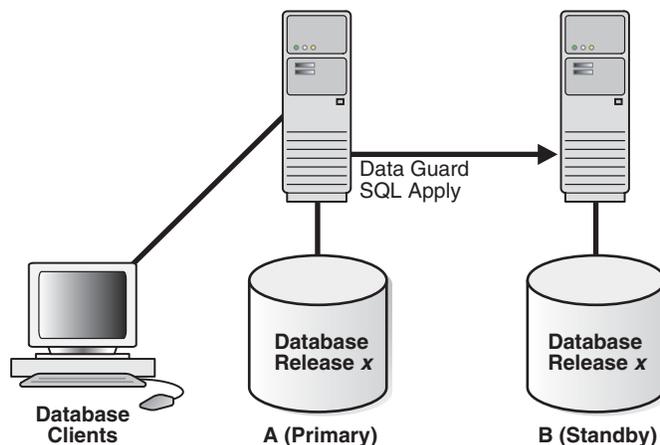
See Also: *Oracle Data Guard Concepts and Administration* for complete information about using the `MANDATORY` and `OPTIONAL` attributes in the `LOG_ARCHIVE_DEST_n` initialization parameter

- The `COMPATIBLE` initialization parameter must match the software release prior to the upgrade. That is, a rolling upgrade from release *x* to release *y* requires that the `COMPATIBLE` initialization parameter be set to release *x* on both the primary and standby databases.

11.3 Figures and Conventions Used in the Upgrade Instructions

Figure 11–1 shows a Data Guard configuration before the upgrade begins, with the primary and logical standby databases both running the same Oracle Database software release.

Figure 11–1 Data Guard Configuration Before Upgrade



During the upgrade process, the Data Guard configuration operates with mixed database releases at several points in this process. Data protection is not available across releases. During these steps, consider having a second standby database in the Data Guard configuration to provide data protection.

The steps and figures describing the upgrade procedure refer to the databases as "Database A" and "Database B" rather than as the "primary database" and "standby database." This is because the databases switch roles during the upgrade procedure. Initially, Database A is the primary database and Database B is the logical standby database, as shown in [Figure 11-1](#).

11.4 Prepare to Upgrade

Perform the following steps to prepare the primary and standby databases for upgrading.

Step 1 Set the COMPATIBLE initialization parameter

Ensure the `COMPATIBLE` initialization parameter specifies the release number for the Oracle Database software running on the primary database prior to the upgrade.

For example, if the primary database is running release 10.1, then set the `COMPATIBLE` initialization parameter to 10.1 on both databases. Be sure to set the `COMPATIBLE` initialization parameter on the standby database first *before* you set it on the primary database.

Step 2 Obtain information about unsupported tables

On Database A, use `DBMS_LOGSTDBY` PL/SQL procedure to capture information about transactions running on the primary database that will not be supported by a logical standby database. Run the following procedures to capture and record the information as events in the `DBA_LOGSTDBY_EVENTS` table:

```
EXEC DBMS_LOGSTDBY.APPLY_SET('MAX_EVENTS_RECORDED', DBMS_LOGSTDBY.MAX_EVENTS);
EXEC DBMS_LOGSTDBY.APPLY_SET('RECORD_UNSUPPORTED_OPERATIONS', 'TRUE');
```

Note: In Oracle Database 10g release 1 (10.1), the `DBMS_LOGSTDBY.MAX_EVENTS` constant was called `DBMS_LOGSTDBY_PUBLIC.MAX_EVENTS`. The effect of the two constants is the same, but in release 2 (10.2) the `DBMS_LOGSTDBY_PUBLIC` package has been eliminated and the definition of the constant moved to the `DBMS_LOGSTDBY` package.

Even though you run these PL/SQL procedures on Database A, they do not affect the primary database. However, by running these procedures on the primary database before creating the logical standby database (and the logical standby database control file), the settings are automatically transferred when you create the logical standby database in step 4.

If a logical standby database already exists (Database B) that can be used for the upgrade procedure, issue the `DBMS_LOGSTDBY` PL/SQL commands on both databases, and then skip to step 3.

See Also: *Oracle Database PL/SQL Packages and Types Reference* for complete information about the `DBMS_LOGSTDBY` procedure

Step 3 Identify unsupported data types and storage attributes

To identify unsupported database objects on the primary database and decide how to handle them, follow these steps:

1. Identify unsupported data types and storage attributes for tables:

- Review the list of supported data types and storage attributes provided in [Appendix C, "Data Type and DDL Support on a Logical Standby Database"](#).
- Query the `DBA_LOGSTDBY_UNUNSUPPORTED` and `DBA_LOGSTDBY_SKIP` views on the primary database. Changes that are made to the listed tables and schemas on the primary database will not be applied on the logical standby database. The following query shows an example of a list of unsupported tables:

```
SQL> SELECT DISTINCT OWNER, TABLE_NAME FROM DBA_LOGSTDBY_UNUNSUPPORTED;
OWNER      TABLE_NAME
-----
OE         CATEGORIES_TAB
OE         CUSTOMERS
OE         WAREHOUSES
PM         ONLINE_MEDIA
PM         PRINT_MEDIA
SCOTT      MYCOMPRESS
SH         MVIEW$_EXCEPTIONS
7 rows selected.
```

```
SQL>
SQL> SELECT OWNER FROM DBA_LOGSTDBY_SKIP
      2  WHERE STATEMENT_OPT = 'INTERNAL SCHEMA';

OWNER
-----
CTXSYS
DBSNMP
DIP
ORDPLUGINS
ORDSYS
OUTLN
SI_INFORMTN_SCHEMA
SYS
SYSTEM
WMSYS
10 rows selected.
```

2. Decide how to handle unsupported tables.

If unsupported objects are being modified on your primary database, it might be possible to perform the upgrade anyway using any of the following methods:

- Temporarily suspend changes to the unsupported tables for the period of time it takes to perform the upgrade procedure.

If you can prevent changes to unsupported changes, then using SQL Apply might still be a viable way to perform the upgrade procedure. This method requires that you prevent users from modifying any unsupported table from the time you create the logical standby control file to the time you complete the upgrade. You can monitor transaction activity in the `DBA_LOGSTDBY_EVENTS` view and discontinue the upgrade (if necessary) up to the time you perform the first switchover.

- Perform the upgrade at a time when users will not be making changes to the unsupported tables.

For logical standby databases that support multiple departments with different requirements, using SQL Apply to perform an upgrade is still possible if you know how users access tables in the database. For example,

assume that the Payroll department updates an object table, but that department updates the database only Monday through Friday. However, the Customer Service department requires database access 24 hours a day, 7 days a week, but uses only supported data types and tables. In this scenario, you could perform the upgrade over a weekend.

If you cannot prevent changes to unsupported tables during the upgrade, any unsupported transactions that occur are recorded in the `DBA_LOGSTDBY_EVENTS` table on the logical standby database. After the upgrade is completed, you might be able to use Oracle Data Pump or the Export/Import utility to import the changed tables to the upgraded databases.

The size of the changed tables will determine how long database operations will be unavailable, so you must decide if a table is too large to export and import its data into the standby database. For example, a 4-terabyte table is not a good candidate for the export/import process.

Note: If you cannot use a logical standby database because the data types in your application are unsupported, then perform the upgrade as documented in *Oracle Database Upgrade Guide*.

Step 4 Create a logical standby database

To create a logical standby database, follow the instructions in [Chapter 4](#).

Oracle recommends configuring a standby redo log on the logical standby database to minimize downtime.

Note: If a logical standby database already exists, go to [Section 11.5, "Upgrade the Databases"](#) to begin the upgrade procedure.

11.5 Upgrade the Databases

This section provides a step-by-step procedure for upgrading the logical standby database and the primary database. [Table 11-1](#) lists the steps.

Table 11-1 Step-by-Step Procedure to Upgrade Oracle Database Software

Step	Description
1	Stop SQL Apply and upgrade the logical standby database
2	Restart SQL Apply
3	Monitor events on the upgraded standby database
4	Begin a switchover
5	Determine if unsupported objects were modified during the upgrade
6	Complete the switchover and activate user applications
7	Upgrade the former primary database
8	Start SQL Apply
9	Optionally, raise the compatibility level on both databases
10	Monitor events on the new logical standby database
11	Optionally, perform another switchover

Note: If your business does not require a logical standby database to support the primary database, you can skip steps 7 through 11.

Step 1 Stop SQL Apply and upgrade the logical standby database

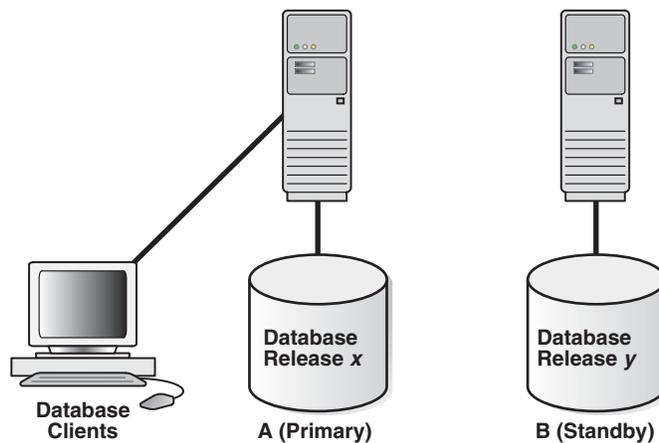
To begin the upgrade, stop SQL Apply and upgrade Oracle database software on the logical standby database (Database B) to release *y*. To stop SQL Apply, issue the following statement on Database B:

```
SQL> ALTER DATABASE STOP LOGICAL STANDBY APPLY;
```

To upgrade Oracle database software, refer to the *Oracle Database Upgrade Guide* for the applicable Oracle Database release.

Figure 11–2 shows Database A running release *x*, and Database B running release *y*. During the upgrade, redo data accumulates on the primary system.

Figure 11–2 Upgrade the Logical Standby Database Release

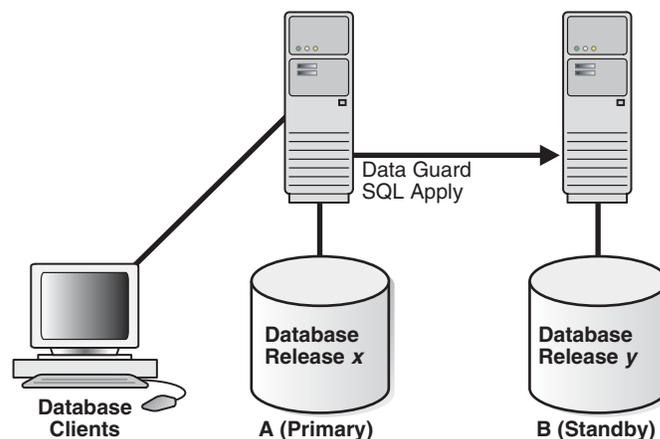


Step 2 Restart SQL Apply

Restart SQL Apply and operate with release *x* on Database A and release *y* on Database B. To start SQL Apply, issue the following statement on Database B:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE;
```

The redo data that was accumulating on the primary system is automatically transmitted and applied on the newly upgraded logical standby database. The Data Guard configuration can run the mixed releases shown in Figure 11–3 for an arbitrary period while you verify that the upgraded Oracle Database software release is running properly in the production environment.

Figure 11–3 Running Mixed Releases

To monitor how quickly Database B is catching up to Database A, query the `V$LOGSTDBY_PROGRESS` view on Database B. For example:

```
SQL> ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-YY HH24:MI:SS';
Session altered.

SQL> SELECT SYSDATE, APPLIED_TIME FROM V$LOGSTDBY_PROGRESS;

SYSDATE          APPLIED_TIME
-----
27-JUN-05 17:07:06 27-JUN-05 17:06:50
```

Step 3 Monitor events on the upgraded standby database

You should frequently query the `DBA_LOGSTDBY_EVENTS` view to learn if there are any DDL and DML statements that have not been applied on Database B.

[Example 11–1](#) demonstrates how monitoring events can alert you to potential differences in the two databases.

Example 11–1 Monitoring Events with `DBA_LOGSTDBY_EVENTS`

```
SQL> SET LONG 1000
SQL> SET PAGESIZE 180
SQL> SET LINESIZE 79
SQL> SELECT EVENT_TIMESTAMP, EVENT, STATUS FROM DBA_LOGSTDBY_EVENTS
ORDER BY EVENT_TIMESTAMP;

EVENT_TIMESTAMP
-----
EVENT
-----
STATUS
-----
...
24-MAY-05 05.18.29.318912 PM
CREATE TABLE SYSTEM.TST (one number)
ORA-16226: DDL skipped due to lack of support

24-MAY-05 05.18.29.379990 PM
"SYSTEM"."TST"
ORA-16129: unsupported dml encountered
```

In the preceding example:

- The ORA-16226 error shows a DDL statement that could not be supported. In this case, it could not be supported because it belongs to an internal schema.
- The ORA-16129 error shows that a DML statement was not applied.

These types of errors indicate that all of the changes that occurred on Database A have not been applied to Database B. At this point, you must decide whether or not to continue with the upgrade procedure. If you are certain that this difference between the logical standby database and the primary database is acceptable, then continue with the upgrade procedure. If not, discontinue and discard Database B and perform the upgrade procedure at another time.

Step 4 Begin a switchover

When you are satisfied that the upgraded database software is operating properly, perform a switchover to reverse the database roles by issuing the following statement on Database A:

```
SQL> ALTER DATABASE COMMIT TO SWITCHOVER TO LOGICAL STANDBY;
```

This may take only a few seconds depending on whether or not the statement must wait for existing transactions to complete. Users still connected to Database A should log off immediately and reconnect to Database B.

Note: If you suspended activity to unsupported tables or packages on Database A when it was the primary database, you must continue to suspend the same activities on Database B while it is the primary database if you eventually plan to switch back to Database A.

Step 5 Determine if unsupported objects were modified during the upgrade

Step 3 "[Monitor events on the upgraded standby database](#)" described how to list unsupported tables that are being modified. If unsupported DML statements were issued on the primary database (as described in [Example 11-1](#)), import the latest version of those tables using an import utility such as Oracle Data Pump.

Note: The tables you import must meet the data type requirements stated in [Section 11.4, "Prepare to Upgrade"](#), step 3.

For example, the following import command truncates the `scott.emp` table and populates it with data matching the former primary database (A):

```
IMPDP SYSTEM/MANAGER NETWORK_LINK=DATABASEA TABLES=SCOTT.EMP TABLE_EXIST_
ACTION=TRUNCATE
```

Step 6 Complete the switchover and activate user applications

When you are satisfied that the upgraded database software is operating properly, complete the switchover to reverse the database roles:

1. On Database B, query the `SWITCHOVER_STATUS` column of the `V$DATABASE` view, as follows:

```
SQL> SELECT SWITCHOVER_STATUS FROM V$DATABASE;
```

```
SWITCHOVER_STATUS
-----
```

TO PRIMARY

- When the `SWITCHOVER_STATUS` column displays `TO PRIMARY`, complete the switchover by issuing the following statement on Database B:

```
SQL> ALTER DATABASE COMMIT TO SWITCHOVER TO LOGICAL PRIMARY;
```

- Activate the user applications and services on Database B, which is now running in the primary database role.

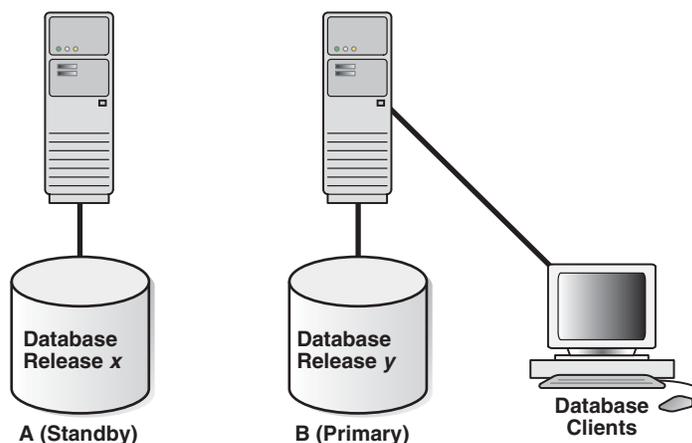
After the switchover, you cannot send redo data from the new primary database (B) that is running the new database software release to the new standby database (A) that is running an older software release. This means the following:

- Redo data is accumulating on the new primary database.
- The new primary database is unprotected at this time.

Figure 11–4 shows Database B, the former standby database (running release *y*), is now the primary database, and Database A, the former primary database (running release *x*), is now the standby database. The users are connected to Database B.

If Database B can adequately serve as the primary database and your business does not require a logical standby database to support the primary database, then you have completed the rolling upgrade process. Allow users to log in to Database B and begin working there, and discard Database A when it is convenient. Otherwise, continue with step 7.

Figure 11–4 After a Switchover

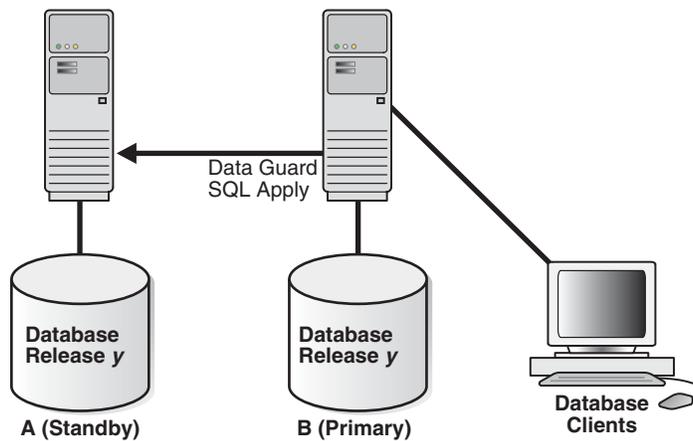


Step 7 Upgrade the former primary database

Database A is still running release *x* and cannot apply redo data from Database B until you upgrade it and start SQL Apply.

For more information about upgrading Oracle Database software, see the *Oracle Database Upgrade Guide* for the applicable Oracle Database release.

Figure 11–5 shows the system after both databases have been upgraded.

Figure 11–5 Both Databases Upgraded**Step 8 Start SQL Apply**

Issue the following statement to start SQL Apply on Database A and, if necessary, create a database link to Database B:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE NEW PRIMARY db_link_to_b;
```

Note: Create a database link prior to issuing this statement *only* if a database link has not already been set up.

When you start SQL Apply on Database A, the redo data that is accumulating on the primary database (B) is sent to the logical standby database (A). The primary database is protected against data loss once all the redo data is available on the standby database.

Step 9 Optionally, raise the compatibility level on both databases

Raise the compatibility level of both databases by setting the `COMPATIBLE` initialization parameter. Set the `COMPATIBLE` parameter on the standby database before you set it on the primary database. See *Oracle Database Reference* for more information about the `COMPATIBLE` initialization parameter.

Step 10 Monitor events on the new logical standby database

To ensure that all changes performed on Database B are properly applied to the logical standby database (A), you should frequently query the `DBA_LOGSTDBY_EVENTS` view, as you did for Database A in step 3. (See [Example 11–1](#).)

If changes were made that invalidate Database A as a copy of your existing primary database, you can discard Database A and create a new logical standby database in its place. See [Chapter 4, "Creating a Logical Standby Database"](#) for complete information.

Step 11 Optionally, perform another switchover

Optionally, perform another switchover of the databases so Database A is once again running in the primary database role (as shown in [Figure 11–1](#)).

See Also: [Section 7.3.1, "Switchovers Involving a Logical Standby Database"](#)

Data Guard Scenarios

This chapter describes scenarios you might encounter while administering your Data Guard configuration. Each scenario can be adapted to your specific environment. [Table 12-1](#) lists the scenarios presented in this chapter.

Table 12-1 *Data Guard Scenarios*

Reference	Scenario
Section 12.1	Setting Up and Verifying Archival Destinations
Section 12.2	Choosing the Best Available Standby Database for a Role Transition
Section 12.3	Configuring a Logical Standby Database to Support a New Primary Database
Section 12.4	Using Flashback Database After a Failover
Section 12.5	Using Flashback Database After Issuing an Open Resetlogs Statement
Section 12.6	Using a Physical Standby Database for Read/Write Testing and Reporting
Section 12.7	Using RMAN Incremental Backups to Roll Forward a Physical Standby Database
Section 12.8	Using a Physical Standby Database with a Time Lag
Section 12.9	Recovering From a Network Failure
Section 12.10	Recovering After the NOLOGGING Clause Is Specified
Section 12.11	Resolving Archive Gaps Manually
Section 12.12	Creating a Standby Database That Uses OMF or ASM

12.1 Setting Up and Verifying Archival Destinations

The following sections set up the `LOG_ARCHIVE_DEST_n` initialization parameter and other related parameters to enable and disable role-specific archiving:

- [Configuring a Primary Database and a Physical Standby Database](#)
- [Configuring a Primary Database and a Logical Standby Database](#)
- [Configuring Both Physical and Logical Standby Databases](#)
- [Verifying the Current VALID_FOR Attribute Settings for Each Destination](#)

12.1.1 Configuring a Primary Database and a Physical Standby Database

[Figure 12-1](#) shows the `chicago` primary database, the `boston` physical standby database, and the initialization parameters for each system.

Figure 12–1 Primary and Physical Standby Databases Before a Role Transition

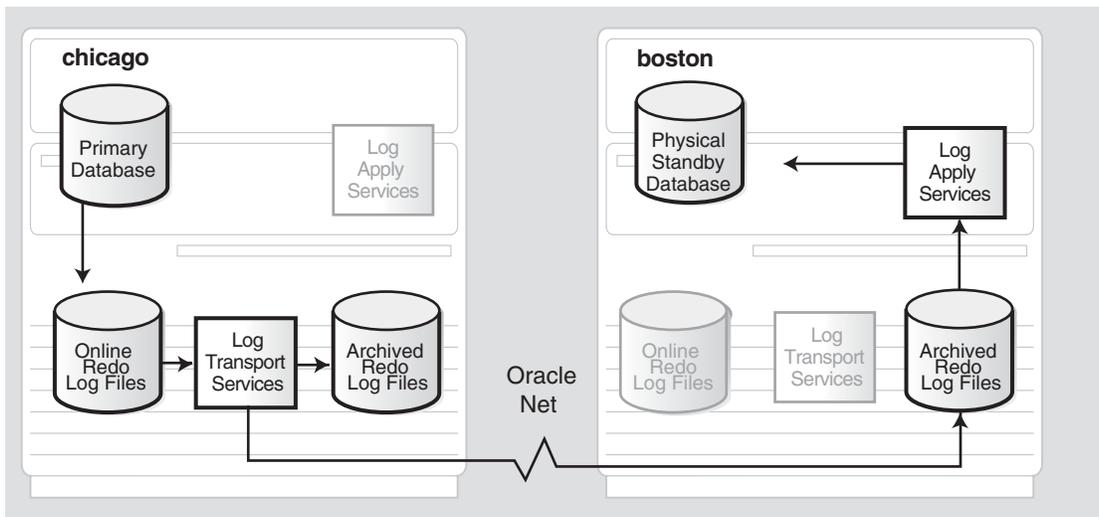


Table 12–2 shows the initialization parameters for the configuration in Figure 12–1.

Table 12–2 Initialization Parameter Settings for Primary and Physical Standby Databases

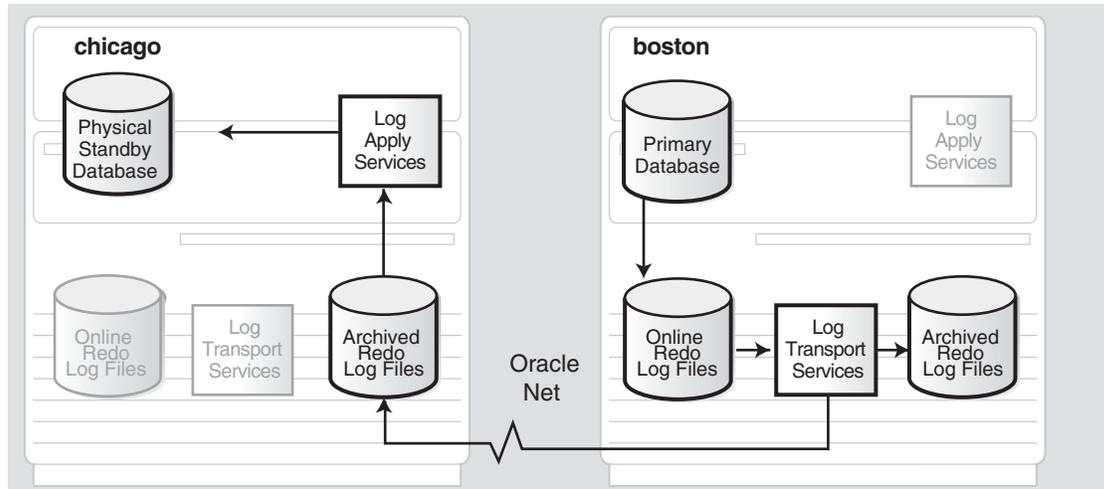
Chicago Database (Primary Role)	Boston Database (Physical Standby Database Role)
DB_UNIQUE_NAME=chicago	DB_UNIQUE_NAME=boston
LOG_ARCHIVE_CONFIG= 'DG_CONFIG=(chicago,boston)'	LOG_ARCHIVE_CONFIG= 'DG_CONFIG=(chicago,boston)'
LOG_ARCHIVE_DEST_1= 'LOCATION=/arch1/chicago/ VALID_FOR=(ALL_LOGFILES,ALL_ROLES) DB_UNIQUE_NAME=boston'	LOG_ARCHIVE_DEST_1= 'LOCATION=/arch1/boston/ VALID_FOR=(ALL_LOGFILES,ALL_ROLES) DB_UNIQUE_NAME=boston'
LOG_ARCHIVE_DEST_2= 'SERVICE=boston VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) DB_UNIQUE_NAME=boston'	LOG_ARCHIVE_DEST_2= 'SERVICE=chicago VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) DB_UNIQUE_NAME=chicago'
LOG_ARCHIVE_DEST_STATE_1=ENABLE	LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_STATE_2=ENABLE	LOG_ARCHIVE_DEST_STATE_2=ENABLE
STANDBY_ARCHIVE_DEST=/arch1/chicago/	STANDBY_ARCHIVE_DEST=/arch1/boston/
REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE	REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE

The following table describes the archival processing shown in Figure 12–1:

	Chicago Database (Primary Role)	Boston Database (Physical Standby Role)
LOG_ARCHIVE_DEST_1	Directs archiving of redo data to the local archived redo log files in /arch1/chicago/.	Directs archiving of redo data to the local archived redo log files in /arch1/boston/.
LOG_ARCHIVE_DEST_2	Directs transmission of the redo data to the remote physical standby database boston.	Is ignored; valid only when boston is running in the primary role.
STANDBY_ARCHIVE_DEST	Is ignored; valid only when chicago is running in the standby role.	Directs archival of redo data to the archived redo log files in the local directory /arch1/boston/.

Figure 12–2 shows the same configuration after a switchover.

Figure 12–2 Primary and Physical Standby Databases After a Role Transition



The following table describes the archival processing shown in [Figure 12–2](#):

	Chicago Database (Physical Standby Role)	Boston Database (Primary Role)
LOG_ARCHIVE_DEST_1	Directs archiving of redo data to the local /arch1/chicago/ directory.	Directs archiving of redo data to the local archived redo log files in /arch1/boston/.
LOG_ARCHIVE_DEST_2	Is ignored; valid only when <i>chicago</i> is running in the primary role.	Directs transmission of redo data to the remote physical standby destination <i>chicago</i> .
STANDBY_ARCHIVE_DEST	Directs archiving of redo data to the archived redo log files in the local directory /arch1/chicago/.	Is ignored; valid only when <i>boston</i> is running in the standby role.

12.1.2 Configuring a Primary Database and a Logical Standby Database

[Figure 12–3](#) shows the *chicago* database running in the primary role, the *denver* database running in the logical standby role, and the initialization parameters for each system. Inactive components are grayed out.

Figure 12-3 Configuring Destinations for a Primary Database and a Logical Standby Database

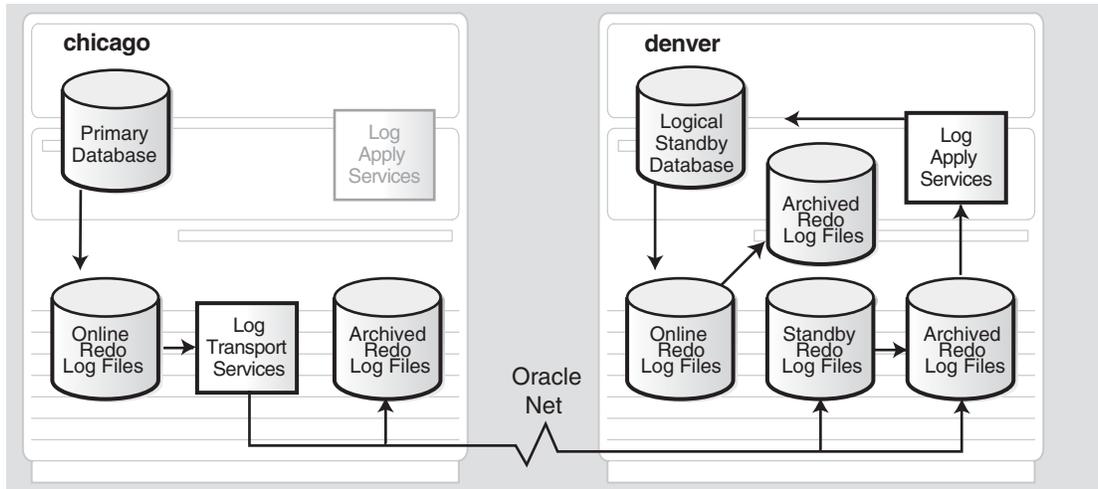


Table 12-3 shows the initialization parameters for the configuration in Figure 12-3.

Table 12-3 Initialization Parameter Settings for Primary and Logical Standby Databases

Chicago Database (Primary Role)	Denver Database (Logical Standby Database Role)
DB_UNIQUE_NAME=chicago	DB_UNIQUE_NAME=denver
LOG_ARCHIVE_CONFIG= 'DG_CONFIG=(chicago,denver)'	LOG_ARCHIVE_CONFIG= 'DG_CONFIG=(chicago,denver)'
LOG_ARCHIVE_DEST_1= 'LOCATION=/arch1/chicago/ VALID_FOR=(ALL_LOGFILES,ALL_ROLES) DB_UNIQUE_NAME=chicago'	LOG_ARCHIVE_DEST_1= 'LOCATION=/arch1/denver/ VALID_FOR=(ONLINE_LOGFILES,ALL_ROLES) DB_UNIQUE_NAME=denver'
LOG_ARCHIVE_DEST_2= 'LOCATION=/arch2/chicago/ VALID_FOR=(STANDBY_LOGFILES,STANDBY_ROLE) DB_UNIQUE_NAME=chicago'	LOG_ARCHIVE_DEST_2= 'LOCATION=/arch2/denver/ VALID_FOR=(STANDBY_LOGFILES,STANDBY_ROLE) DB_UNIQUE_NAME=denver'
LOG_ARCHIVE_DEST_3= 'SERVICE=denver VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) DB_UNIQUE_NAME=denver'	LOG_ARCHIVE_DEST_3= 'SERVICE=chicago VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) DB_UNIQUE_NAME=chicago'
LOG_ARCHIVE_DEST_STATE_1=ENABLE	LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_STATE_2=ENABLE	LOG_ARCHIVE_DEST_STATE_2=ENABLE
LOG_ARCHIVE_DEST_STATE_3=ENABLE	LOG_ARCHIVE_DEST_STATE_3=ENABLE
STANDBY_ARCHIVE_DEST=/arch2/chicago/	STANDBY_ARCHIVE_DEST=/arch2/denver/
REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE	REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE

The following table describes the archival processing shown in Figure 12-3:

Chicago Database (Primary Role)	Denver Database (Logical Standby Role)
LOG_ARCHIVE_DEST_1 Directs archiving of redo data generated by the primary database from the local online redo log files to the local archived redo log files in /arch1/chicago/.	LOG_ARCHIVE_DEST_1 Directs archiving of redo data generated by the logical standby database from the local online redo log files to the local archived redo log files in /arch1/denver/.

	Chicago Database (Primary Role)	Denver Database (Logical Standby Role)
LOG_ARCHIVE_DEST_2	Is ignored; valid only when <code>chicago</code> is running in the standby role. (You must configure a standby redo log on this site to perform switchovers.)	Directs archiving of redo data from the standby redo log files to the local archived redo log files in <code>/arch2/denver/</code> .
LOG_ARCHIVE_DEST_3	Directs transmission of redo data to the remote logical standby destination <code>denver</code> .	Is ignored; valid only when <code>denver</code> is running in the primary role.
STANDBY_ARCHIVE_DEST	Is ignored; valid only when <code>chicago</code> is running in the standby role.	Directs archiving of redo data received from the primary database directly to archived redo log files in <code>/arch2/denver/</code> .

Unlike physical standby databases, logical standby databases are open databases that generate redo data and have multiple log files (online redo log files, archived redo log files, and standby redo log files). It is good practice to specify separate local destinations for:

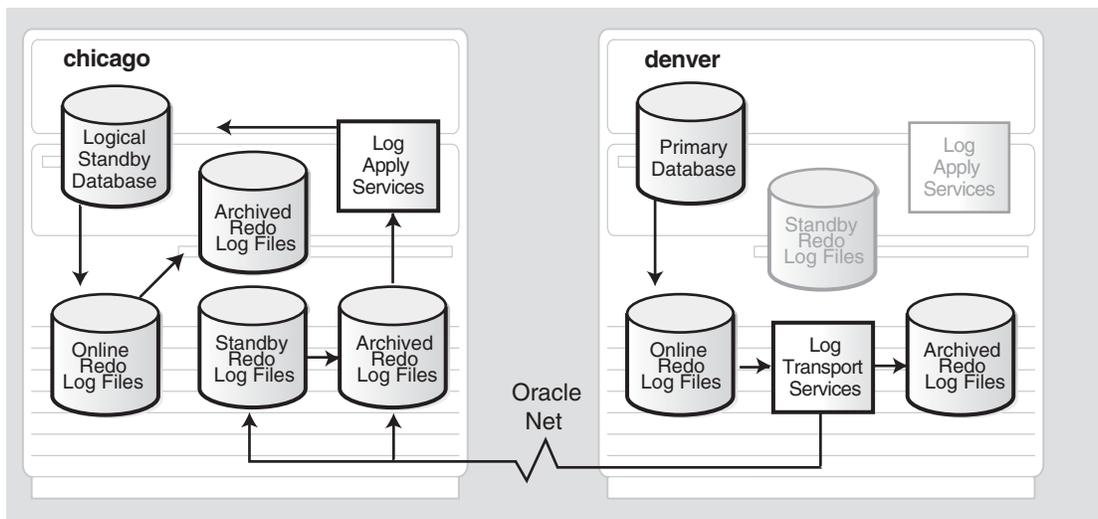
- Archived redo log files that store redo data *generated by the logical standby database*. In [Figure 12-3](#), this is configured as the `LOG_ARCHIVE_DEST_1=LOCATION=/arch1/denver` destination.
- Archived redo log files that store redo data *received from the primary database*. In [Figure 12-3](#), this is configured as the `LOG_ARCHIVE_DEST_2=LOCATION=/arch2/denver` destination.

In [Figure 12-3](#), the `STANDBY_ARCHIVE_DEST` parameter is configured to the same location for these purposes:

- If the standby redo log files fill up, redo data received from the primary database is archived directly to the archived redo log files in this location (described in [Section 5.7.1](#)).
- If there is an archive gap, archived redo log files retrieved from other databases are copied to this location (described in [Section 5.8](#)).

Because the example configurations shown in [Figure 12-3](#) (and [Figure 12-4](#)) do not include a physical standby database, the configuration sets up the `LOG_ARCHIVE_DEST_3` destination for switchover with the logical standby database. [Figure 12-4](#) shows the same configuration after a switchover.

Figure 12–4 Primary and Logical Standby Databases After a Role Transition



The following table describes the archival processing shown in [Figure 12–4](#):

	Chicago Database (Logical Standby Role)	Denver Database (Primary Role)
LOG_ARCHIVE_DEST_1	Directs archiving of redo data generated by the logical standby database from the local online redo log files to the local archived redo log files in /arch1/chicago/.	Directs archiving of redo data from the local online redo log files to the local archived redo log files in /arch1/denver/.
LOG_ARCHIVE_DEST_2	Directs archiving of redo data from the standby redo log files to the archived redo log file in /arch2/chicago/.	Is ignored; valid only when denver is running in the standby role.
LOG_ARCHIVE_DEST_3	Is ignored; valid only when chicago is running in the primary role.	Directs transmission of redo data to the remote logical standby destination chicago.
STANDBY_ARCHIVE_DEST	Directs archiving of the redo data received from the primary database directly to the archived redo log files in /arch2/chicago/.	Is ignored; valid only when denver is running in the standby role.

12.1.3 Configuring Both Physical and Logical Standby Databases

[Figure 12–5](#) shows the `chicago` database running in the primary role, the `boston` database running in the physical standby role, and the `denver` database running in the logical standby database role. The initialization parameters are shown under each system. Components that are grayed out are inactive for the database’s current role. This example assumes that a switchover would occur only between `chicago` and `boston`. In this configuration, the `denver` logical standby database is intended to be a reporting database only; `denver` will never be the target of a switchover or run in the primary database role.

Figure 12–5 Configuring a Primary Database with Physical and Logical Standby Databases

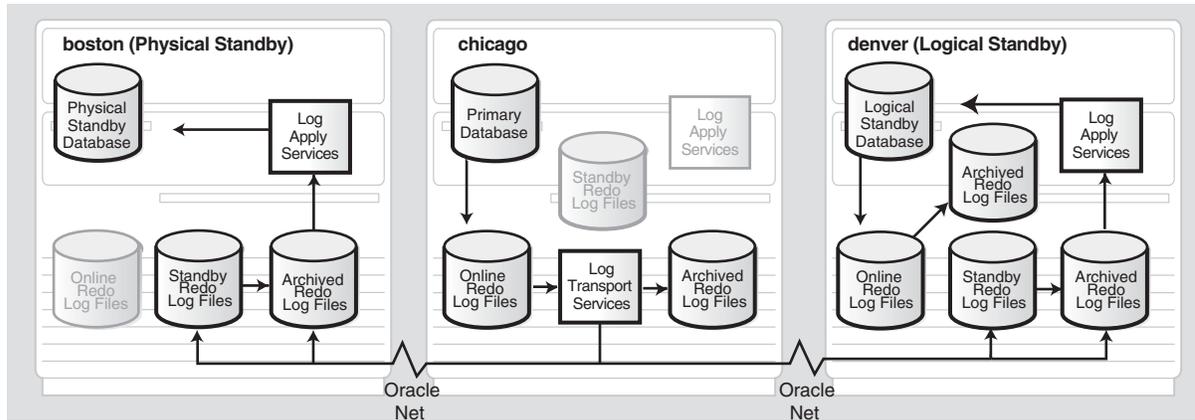


Table 12–4 shows the initialization parameters for the databases in Figure 12–5.

Table 12–4 Initialization Parameters for Primary, Physical, and Logical Standby Databases

Boston Database (Standby Role)	Chicago Database (Primary Role)	Denver Database (Standby Role)
DB_UNIQUE_NAME=boston	DB_UNIQUE_NAME=chicago	DB_UNIQUE_NAME=denver
LOG_ARCHIVE_CONFIG='DG_CONFIG=(chicago,boston,denver)'	LOG_ARCHIVE_CONFIG='DG_CONFIG=(chicago,boston,denver)'	LOG_ARCHIVE_CONFIG='DG_CONFIG=(chicago,boston,denver)'
LOG_ARCHIVE_DEST_1='LOCATION=/arch1/boston/VALID_FOR=(ONLINE_LOGFILES,ALL_ROLES)DB_UNIQUE_NAME=boston'	LOG_ARCHIVE_DEST_1='LOCATION=/arch1/chicago/VALID_FOR=(ONLINE_LOGFILES,ALL_ROLES)DB_UNIQUE_NAME=chicago'	LOG_ARCHIVE_DEST_1='LOCATION=/arch1/denver/VALID_FOR=(ONLINE_LOGFILES,ALL_ROLES)DB_UNIQUE_NAME=denver'
LOG_ARCHIVE_DEST_2='SERVICE=denverVALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)DB_UNIQUE_NAME=denver'	LOG_ARCHIVE_DEST_2='SERVICE=denverVALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)DB_UNIQUE_NAME=denver'	LOG_ARCHIVE_DEST_2='SERVICE=denverVALID_FOR=(ONLINE_LOGFILES,STANDBY_ROLE)DB_UNIQUE_NAME=denver'
LOG_ARCHIVE_DEST_3='SERVICE=chicagoVALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)DB_UNIQUE_NAME=chicago'	LOG_ARCHIVE_DEST_3='SERVICE=bostonVALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)DB_UNIQUE_NAME=boston'	LOG_ARCHIVE_DEST_3='SERVICE=chicagoVALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)DB_UNIQUE_NAME=boston'
LOG_ARCHIVE_DEST_STATE_1=ENABLE	LOG_ARCHIVE_DEST_STATE_1=ENABLE	LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_STATE_2=ENABLE	LOG_ARCHIVE_DEST_STATE_2=ENABLE	LOG_ARCHIVE_DEST_STATE_2=ENABLE
LOG_ARCHIVE_DEST_STATE_3=ENABLE	LOG_ARCHIVE_DEST_STATE_3=ENABLE	LOG_ARCHIVE_DEST_STATE_3=ENABLE
STANDBY_ARCHIVE_DEST=/arch1/boston/REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE	STANDBY_ARCHIVE_DEST=/arch1/chicago/REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE	STANDBY_ARCHIVE_DEST=/arch2/denver/REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE

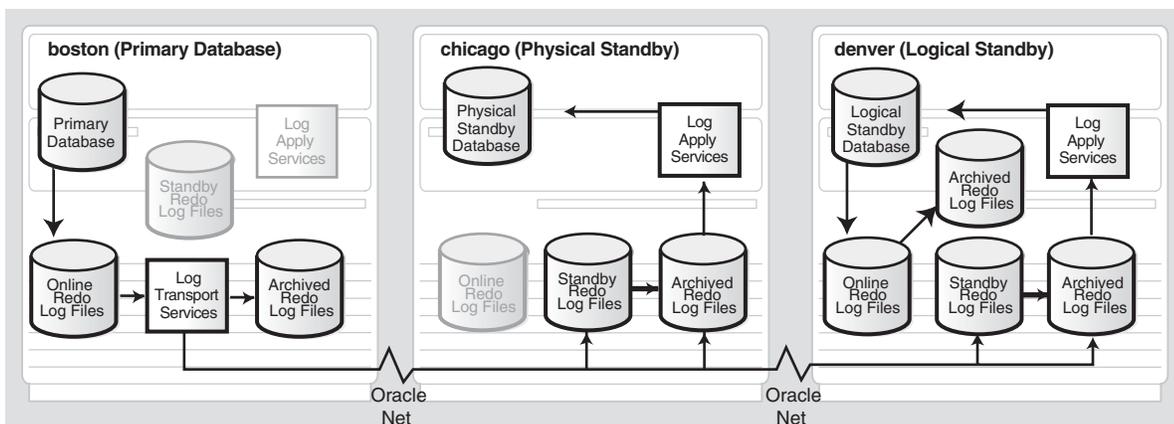
The following table describes the archival processing shown in Figure 12–5:

	Chicago Database (Primary Role)	Boston Database (Standby Role)	Denver Database (Standby Role)
LOG_ARCHIVE_DEST_1	Directs archiving of redo data from the online redo log files to the local archived redo log files in /arch1/chicago/.	Directs archiving of redo data from the standby redo log files to the local archived redo log files in /arch1/boston/.	Directs archiving of redo data generated by the logical standby database from the local online redo log files to the local archived redo log files in /arch1/denver/.
LOG_ARCHIVE_DEST_2	Directs transmission of redo data to the remote logical standby destination denver.	Is ignored; valid only when boston is running in the primary role.	Directs archiving of redo data from the standby redo log files to the local archived redo log files in /arch2/denver/.

	Chicago Database (Primary Role)	Boston Database (Standby Role)	Denver Database (Standby Role)
LOG_ARCHIVE_DEST_3	Directs transmission of redo data to the remote physical standby destination <code>boston</code> .	Is ignored; valid only when <code>boston</code> is running in the primary role.	Is not defined for this database.
STANDBY_ARCHIVE_DEST	Is ignored; valid only for standby role.	Directs archiving of redo data received from the primary database directly to archived redo log files in <code>/arch1/boston/</code> .	Directs archiving of redo data received from the primary database directly to archived redo log files in <code>/arch2/denver/</code> .

Figure 12-6 shows the same configuration after a switchover changes the `chicago` database to the standby role and the `boston` database to the primary role.

Figure 12-6 Primary, Physical, and Logical Standby Databases After a Role Transition



The following table describes the archival processing shown in Figure 12-6:

	Chicago Database (Standby Role)	Boston Database (Primary Role)	Denver Database (Standby Role)
LOG_ARCHIVE_DEST_1	Directs archival of redo data from the standby redo log files to the local archived redo log files in <code>/arch1/chicago/</code> .	Directs archival of redo data from the online redo log files to the local archived redo log files in <code>/arch1/boston/</code> .	Directs archival of redo data generated by the logical standby database from the local online redo log files to the local archived redo log files in <code>/arch1/denver/</code> .
LOG_ARCHIVE_DEST_2	Is ignored; valid only when <code>chicago</code> is running in the primary role.	Directs transmission of redo data to the remote logical standby destination <code>denver</code> .	Directs archival of redo data from the standby redo log files to the local archived redo log files in <code>/arch2/denver/</code> .
LOG_ARCHIVE_DEST_3	Is ignored; valid only when <code>chicago</code> is running in the primary role.	Directs transmission of redo data to the remote physical standby destination <code>chicago</code> .	Is not defined for this database.
STANDBY_ARCHIVE_DEST	Directs archival of redo data received from the primary database directly to the archived redo log files in <code>/arch1/chicago/</code> .	Is ignored; valid only for standby role.	Directs archival of redo data received from the primary database directly to archived redo log files in <code>/arch2/denver/</code> .

12.1.4 Verifying the Current VALID_FOR Attribute Settings for Each Destination

To see whether or not the current VALID_FOR attribute settings are valid *right now* for each destination in the Data Guard configuration, query the V\$ARCHIVE_DEST view, as shown in [Example 12-1](#).

Example 12-1 Finding VALID_FOR Information in the V\$ARCHIVE_DEST View

```
SQL> SELECT DEST_ID,VALID_TYPE,VALID_ROLE,VALID_NOW FROM V$ARCHIVE_DEST;
DEST_ID  VALID_TYPE          VALID_ROLE  VALID_NOW
-----  -
1        ALL_LOGFILES       ALL_ROLES   YES
2        STANDBY_LOGFILE   STANDBY_ROLE  WRONG VALID_TYPE
3        ONLINE_LOGFILE    STANDBY_ROLE  WRONG VALID_ROLE
4        ALL_LOGFILES       ALL_ROLES   UNKNOWN
5        ALL_LOGFILES       ALL_ROLES   UNKNOWN
6        ALL_LOGFILES       ALL_ROLES   UNKNOWN
7        ALL_LOGFILES       ALL_ROLES   UNKNOWN
8        ALL_LOGFILES       ALL_ROLES   UNKNOWN
9        ALL_LOGFILES       ALL_ROLES   UNKNOWN
10       ALL_LOGFILES       ALL_ROLES   UNKNOWN
10 rows selected.
```

In [Example 12-1](#), each line represents one of the ten destinations in the Data Guard configuration. The first line indicates that the VALID_FOR attribute for LOG_ARCHIVE_DEST_1 is set to (ALL_LOGFILES, ALL_ROLES), which is the only keyword pair that is valid at all times.

More interesting are the second and third lines in the view, which are both currently invalid, but for different reasons:

- LOG_ARCHIVE_DEST_2 is set to (STANDBY_LOGFILES, STANDBY_ROLE), but the WRONG VALID_TYPE is returned because this standby destination does not have a standby redo log implemented.
- LOG_ARCHIVE_DEST_3 is set to (ONLINE_LOGFILES, STANDBY_ROLE), but the WRONG VALID_ROLE is returned because this destination is currently running in the primary database role.

All of the other destinations are shown as UNKNOWN, which indicates the destinations are either undefined or the database is started and mounted but archiving is not currently taking place. See the V\$ARCHIVE_DEST view in the *Oracle Database Reference* for information about these and other columns.

12.2 Choosing the Best Available Standby Database for a Role Transition

The following sections provide scenarios that illustrate, in a step-by-step fashion, how to choose the best available standby database for failover:

- [Example: Best Physical Standby Database for a Failover](#)
- [Example: Best Logical Standby Database for a Failover](#)

See Also: [Section 7.1.2](#) for general guidelines about selecting a target standby database for switchover and failover

If a configuration contains both physical and logical standby databases, Oracle recommends that you perform the role transition using the best available physical standby database. This is recommended because:

- A logical standby database might contain only a subset of the data present in the primary database.
- A role transition involving a logical standby database requires that any existing physical standby databases be re-created from a copy of the new primary database (after the role transition is complete) to continue to participate in the Data Guard configuration.

Because of these limitations, a logical standby database should be considered as the target for a role transition only in the following special situations:

- The configuration contains only logical standby databases.
- It is critical to fail over a standby database to the primary role as quickly as possible, and the most current logical standby database in the configuration is significantly more current than the most current physical standby database in the configuration.

Once you determine whether to use a physical or a logical standby database, the specific standby database you select as the target for the role transition is determined by how much of the recent primary database modifications are available on the standby database. Because the primary database remains accessible during switchovers, there will be no loss of data, and the choice of the standby database used during a switchover will only affect the time required to complete the switchover. For failovers, however, the choice of standby database might involve trade-off between additional risk of data loss and the time required to transition a standby database to the primary role.

12.2.1 Example: Best Physical Standby Database for a Failover

In a disaster, the most critical task for the DBA is to determine if it is quicker and safer to repair the primary database or fail over to a standby database. When deciding that a failover is necessary and multiple physical standby databases are configured, the DBA must choose which physical standby database is the best target for the failover. While there are many environmental factors that can affect which standby database represents the best choice, this scenario assumes these things to be equal for emphasizing data loss assessment.

This scenario begins with a Data Guard configuration consisting of the HQ primary database and two physical standby databases, SAT and NYC. The HQ database is operating in maximum availability protection mode, and the standby databases are each configured with three standby redo log files. See [Section 1.4](#) for more information about the maximum availability protection mode for physical standby databases.

[Table 12-5](#) provides information about the databases used in this scenario.

Table 12-5 Identifiers for the Physical Standby Database Example

Identifier	HQ Database	SAT Database	NYC Database
Location	San Francisco	Seattle	New York City
Database name	HQ	HQ	HQ
Instance name	HQ	SAT	NYC
Initialization parameter file	hq_init.ora	sat_init.ora	nyc_init.ora
Control file	hq_cf1.f	sat_cf1.f	nyc_cf1.f
Datafile	hq_db1.f	sat_db1.f	nyc_db1.f
Redo log file 1	hq_log1.f	sat_log1.f	nyc_log1.f

Table 12–5 (Cont.) Identifiers for the Physical Standby Database Example

Identifier	HQ Database	SAT Database	NYC Database
Redo log file 2	hq_log2.f	sat_log2.f	nyc_log2.f
Standby redo log file 1	hq_srl1.f	sat_srl1.f	nyc_srl1.f
Standby redo log file 2	hq_srl2.f	sat_srl2.f	nyc_srl2.f
Standby redo log file 3	hq_srl3.f	sat_srl3.f	nyc_srl3.f
Primary protection mode	Maximum availability	Not applicable	Not applicable
Standby protection mode	Not applicable	Maximum availability (synchronous)	Maximum performance (asynchronous)
Network service name (client defined)	hq_net	sat_net	nyc_net
Listener	hq_listener	sat_listener	nyc_listener

Note: The New York city database is operating in maximum performance mode because sending redo data synchronously from HQ to NYC might impact the primary database performance during peak workload periods. However, the New York City standby database is still considered a viable candidate for failovers because it uses a standby redo log.

Assume that an event occurs in San Francisco where the primary site is located, and the primary site is damaged in such a way that it cannot be repaired in a timely manner. You must fail over to one of the standby databases. You cannot assume that the DBA who set up the multiple standby database configuration is available to decide to which standby database to fail over. Therefore, it is imperative to have a disaster recovery plan at each standby site, as well as at the primary site. Each member of the disaster recovery team needs to know about the disaster recovery plan and be aware of the procedures to follow. This scenario identifies the information you need when deciding which standby database should be the target of the failover.

One method of conveying information to the disaster recovery team is to include a ReadMe file at each standby site. This ReadMe file is created and maintained by the DBA and should describe how to:

- Log on to the local Oracle database as a DBA
- Log on to each system where the standby databases are located
- Get instructions for going through firewalls, because there might be firewalls between systems
- Log on to other Oracle databases as a DBA
- Identify the most up-to-date standby database
- Perform the standby database failover
- Configure network settings to ensure client applications access the new primary database, instead of the original primary database

When choosing a standby database, there are two critical considerations: which standby database received the most recent redo data and which standby database has applied the most redo.

Follow these steps to determine which standby database is the best candidate for failover when only physical standby databases are in the configuration. Always start with the standby database providing the highest protection level. In this scenario, the Seattle standby database provides the highest protection level because it is operating in maximum availability protection mode.

Step 1 Connect to the SAT physical standby database.

Issue a SQL statement such as the following:

```
SQL> CONNECT SYS/CHANGE_ON_INSTALL AS SYSDBA;
```

Step 2 Determine how much current redo data is available in the archived redo log file.

Query the columns in the V\$MANAGED_STANDBY view, as shown:

```
SQL> SELECT THREAD#, SEQUENCE#, BLOCK#, BLOCKS
2> FROM V$MANAGED_STANDBY WHERE STATUS='RECEIVING';
```

THREAD#	SEQUENCE#	BLOCK#	BLOCKS
1	14	234	16

This standby database received 249 blocks of redo data from the primary database. To compute the number of blocks received, add the BLOCKS column value to the BLOCK# column value, and subtract 1 (because block number 234 is included in the 16 blocks received).

Note: Depending on how long the primary database has been unavailable, the previous query might not return any selected rows because the RFS process might detect the network disconnection and terminate itself. If this occurs, it is always best to select a standby database that is configured to receive the redo data in a synchronous manner.

Step 3 Obtain a list of the archived redo log files that were applied or are currently pending application to the SAT database.

Query the V\$ARCHIVED_LOG view:

```
SQL> SELECT SUBSTR(NAME,1,25) FILE_NAME, SEQUENCE#, APPLIED
2> FROM V$ARCHIVED_LOG ORDER BY SEQUENCE#;
```

FILE_NAME	SEQUENCE#	APP
/oracle/dbs/hq_sat_2.log	2	YES
/oracle/dbs/hq_sat_3.log	3	YES
/oracle/dbs/hq_sat_4.log	4	YES
/oracle/dbs/hq_sat_5.log	5	YES
/oracle/dbs/hq_sat_6.log	6	YES
/oracle/dbs/hq_sat_7.log	7	YES
/oracle/dbs/hq_sat_8.log	8	YES
/oracle/dbs/hq_sat_9.log	9	YES
/oracle/dbs/hq_sat_10.log	10	YES
/oracle/dbs/hq_sat_11.log	11	YES
/oracle/dbs/hq_sat_13.log	13	NO

This output indicates that archived redo log file 11 was completely applied to the standby database. (The line for log file 11 in the example output is in bold typeface to assist you in reading the output. The actual output will not display bolding.)

Also, notice the gap in the sequence numbers in the `SEQUENCE#` column. In the example, the gap indicates the SAT standby database is missing archived redo log file number 12.

Step 4 Connect to the NYC database to determine if it is more recent than the SAT standby database.

Issue a SQL statement such as the following:

```
SQL> CONNECT SYS/CHANGE_ON_INSTALL AS SYSDBA;
```

Step 5 Determine how much current redo data is available in the archived redo log file.

Query the columns in the `V$MANAGED_STANDBY` view as shown:

```
SQL> SELECT THREAD#, SEQUENCE#, BLOCK#, BLOCKS
2> FROM V$MANAGED_STANDBY WHERE STATUS='RECEIVING';
```

THREAD#	SEQUENCE#	BLOCK#	BLOCKS
1	14	157	93

This standby database has also received 249 blocks of redo information from the primary database. To compute the number of blocks received, add the `BLOCKS` column value to the `BLOCK#` column value, and subtract 1 (because block number 157 is included in the 93 blocks received).

Step 6 Obtain a list of the archived redo log files that were applied or are currently pending application to the NYC database.

Query the `V$ARCHIVED_LOG` view:

```
SQL> SELECT SUBSTR(NAME,1,25) FILE_NAME, SEQUENCE#, APPLIED
2> FROM V$ARCHIVED_LOG ORDER BY SEQUENCE#;
```

FILE_NAME	SEQUENCE#	APP
/oracle/dbs/hq_nyc_2.log	2	YES
/oracle/dbs/hq_nyc_3.log	3	YES
/oracle/dbs/hq_nyc_4.log	4	YES
/oracle/dbs/hq_nyc_5.log	5	YES
/oracle/dbs/hq_nyc_6.log	6	YES
/oracle/dbs/hq_nyc_7.log	7	YES
/oracle/dbs/hq_nyc_8.log	8	NO
/oracle/dbs/hq_nyc_9.log	9	NO
/oracle/dbs/hq_nyc_10.log	10	NO
/oracle/dbs/hq_nyc_11.log	11	NO
/oracle/dbs/hq_nyc_12.log	12	NO
/oracle/dbs/hq_nyc_13.log	13	NO

This output indicates that archived redo log file 7 was completely applied to the standby database. (The line for log file 7 in the example output is in bold typeface to assist you in reading the output. The actual output will not display bolding.)

More redo data was received at this location, but less was applied to the standby database.

Step 7 Choose the best target standby database.

In most cases, the physical standby database you choose as a failover target should provide a balance between risk of data loss and time required to perform the role transition. As you analyze this information to make a decision about the best failover candidate in this scenario, consider the following:

- For minimal risk of data loss during a failover, you should choose the NYC database as the best target standby database because Steps 5 and 6 revealed that the NYC site has the most recoverable redo.
- For minimal primary database downtime during the failover operation, you should choose the SAT database as the best target standby database. This database is a more appropriate candidate because the queries in Steps 2 through 6 reveal that the SAT database applied 5 archived redo log files more than the NYC database. However, if it is not possible to obtain and apply a copy of the missing archived redo log file (log 12 in the example), then you will not be able to make the SAT database as current as you can the NYC database. Therefore, you will lose the unapplied data (log files 12, 13, and part of log file 14 in the example).

Based on your business requirements, choose the best target standby database.

Step 8 Bring the selected standby database to its most current state.

- **If you chose the SAT database as the best target based on your business requirements, perform the following steps:**
 1. Retrieve any missing archived redo log files using an operating system copy utility. (This example uses the UNIX `cp` command). In this case, the SAT database is missing archived redo log file 12. Because the NYC database received this archived redo log file, you can copy it from the NYC database to the SAT database, as follows:

```
% cp /net/nyc/oracle/dbs/hq_nyc_12.log /net/sat/oracle/dbs/hq_sat_12.log
```

2. Determine if a partial archived redo log file exists for the next sequence number. In this example, the next sequence number should be 14. The following UNIX command searches the directory on the SAT database for the presence of an archived redo log file named `hq_sat_14.log`:

```
% ls -l /net/sat/oracle/dbs/hq_sat_14.log
/net/sat/oracle/dbs/hq_sat_14.log: No such file or directory
```

Because the SAT standby database is using standby redo log files, there should not be any partial archived redo log files.

3. Register the retrieved archived redo log file. (There is no need to stop log apply services).

```
SQL> ALTER DATABASE REGISTER PHYSICAL LOGFILE '/oracle/dbs/hq_sat_12.log';
```

4. Query the `V$ARCHIVED_LOG` view again to make sure the archived redo log files were successfully applied:

```
SQL> SELECT SUBSTR(NAME,1,25) FILE_NAME, SEQUENCE#, APPLIED
2> FROM V$ARCHIVED_LOG ORDER BY SEQUENCE#;
```

FILE_NAME	SEQUENCE#	APP
/oracle/dbs/hq_sat_2.log	2	YES
/oracle/dbs/hq_sat_3.log	3	YES
/oracle/dbs/hq_sat_4.log	4	YES
/oracle/dbs/hq_sat_5.log	5	YES

```

/oracle/dbs/hq_sat_6.log          6 YES
/oracle/dbs/hq_sat_7.log          7 YES
/oracle/dbs/hq_sat_8.log          8 YES
/oracle/dbs/hq_sat_9.log          9 YES
/oracle/dbs/hq_sat_10.log         10 YES
/oracle/dbs/hq_sat_11.log         11 YES
/oracle/dbs/hq_sat_12.log         12 YES
/oracle/dbs/hq_sat_13.log         13 YES

```

■ **If you chose the NYC database as the best target based on your business requirements, perform the following steps:**

1. Determine if a partial archived redo log file exists for the next sequence number. The following UNIX command searches the directory on the NYC database for the presence of an archived redo log file named with the next sequence (hq_nyc_14):

```

% ls -l /net/nyc/oracle/dbs/hq_nyc_14.log
/net/nyc/oracle/dbs/hq_nyc_14.log: No such file or directory

```

Because the NYC standby database is using standby redo log files, there should not be any partial archived redo log files.

2. Start log apply services to apply the most current log file:

```

SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE
2> DISCONNECT FROM SESSION;

```

3. Query the V\$ARCHIVED_LOG view again to make sure the archived redo log files were successfully applied:

```

SQL> SELECT SUBSTR(NAME,1,25) FILE_NAME, SEQUENCE#, APPLIED
2> FROM V$ARCHIVED_LOG ORDER BY SEQUENCE#;

```

FILE_NAME	SEQUENCE#	APP
/oracle/dbs/hq_nyc_2.log	2	YES
/oracle/dbs/hq_nyc_3.log	3	YES
/oracle/dbs/hq_nyc_4.log	4	YES
/oracle/dbs/hq_nyc_5.log	5	YES
/oracle/dbs/hq_nyc_6.log	6	YES
/oracle/dbs/hq_nyc_7.log	7	YES
/oracle/dbs/hq_nyc_8.log	8	YES
/oracle/dbs/hq_nyc_9.log	9	YES
/oracle/dbs/hq_nyc_10.log	10	YES
/oracle/dbs/hq_nyc_11.log	11	YES
/oracle/dbs/hq_nyc_12.log	12	NO
/oracle/dbs/hq_nyc_13.log	13	NO

Applying the archived redo log files might take some time to complete. Therefore, you must wait until all archived redo log files are designated as applied, as shown:

```

SQL> SELECT SUBSTR(NAME,1,25) FILE_NAME, SEQUENCE#, APPLIED
2> FROM V$ARCHIVED_LOG ORDER BY SEQUENCE#;

```

FILE_NAME	SEQUENCE#	APP
/oracle/dbs/hq_nyc_2.log	2	YES
/oracle/dbs/hq_nyc_3.log	3	YES
/oracle/dbs/hq_nyc_4.log	4	YES
/oracle/dbs/hq_nyc_5.log	5	YES

```

/oracle/dbs/hq_nyc_6.log          6 YES
/oracle/dbs/hq_nyc_7.log          7 YES
/oracle/dbs/hq_nyc_8.log          8 YES
/oracle/dbs/hq_nyc_9.log          9 YES
/oracle/dbs/hq_nyc_10.log         10 YES
/oracle/dbs/hq_nyc_11.log         11 YES
/oracle/dbs/hq_nyc_12.log         12 YES
/oracle/dbs/hq_nyc_13.log         13 YES

```

Step 9 Perform the failover.

You are now ready to stop log apply services and fail over the selected physical standby database to the primary role.

See [Section 7.2.2](#) for additional information about how to fail over to a physical standby database.

12.2.2 Example: Best Logical Standby Database for a Failover

In a disaster when only logical standby databases are available, the critical task is to determine which logical standby database is the best target for the failover. While there are many environmental factors that can affect which is the best target standby database, this scenario assumes these things to be equal for emphasizing data loss assessment. See [Section 1.4](#) for more information about the maximum availability protection mode for logical standby databases.

This scenario starts out with a Data Guard configuration consisting of the HQ primary database and two logical standby databases, SAT and NYC. [Table 12–6](#) provides information about each of these databases.

Table 12–6 Identifiers for Logical Standby Database Example

Identifier	HQ Database	SAT Database	NYC Database
Location	San Francisco	Seattle	New York City
Database name	HQ	SAT	NYC
Instance name	HQ	SAT	NYC
Initialization parameter file	hq_init.ora	sat_init.ora	nyc_init.ora
Control file	hq_cf1.f	sat_cf1.f	nyc_cf1.f
Datafile	hq_db1.f	sat_db1.f	nyc_db1.f
Redo log file 1	hq_log1.f	sat_log1.f	nyc_log1.f
Redo log file 2	hq_log2.f	sat_log2.f	nyc_log2.f
Database link (client-defined)	hq_link	sat_link	nyc_link
Network service name (client-defined)	hq_net	sat_net	nyc_net
Listener	hq_listener	sat_listener	nyc_listener

Follow these steps to determine which standby database is the best candidate for failover when only logical standby databases are in the configuration:

Step 1 Connect to the SAT logical standby database.

Issue a SQL statement such as the following:

```
SQL> CONNECT SYS/CHANGE_ON_INSTALL AS SYSDBA;
```

Step 2 Determine the highest applied SCN and highest (newest) applicable SCN on the SAT database.

Query the following columns in the V\$LOGSTDBY_PROGRESS view:

```
SQL> SELECT APPLIED_SCN, LATEST_SCN FROM V$LOGSTDBY_PROGRESS;
```

```
APPLIED_SCN LATEST_SCN
-----
144059      144059
```

Step 3 Obtain a list of the archived redo log files that were applied or are currently pending application to the SAT database.

Query the DBA_LOGSTDBY_LOG view:

```
SQL> SELECT SUBSTR(FILE_NAME,1,25) FILE_NAME, SUBSTR(SEQUENCE#,1,4) "SEQ#",
2> FIRST_CHANGE#, NEXT_CHANGE#, TO_CHAR(TIMESTAMP, 'HH:MI:SS') TIMESTAMP,
3> DICT_BEGIN BEG, DICT_END END, SUBSTR(THREAD#,1,4) "THR#"
4> FROM DBA_LOGSTDBY_LOG ORDER BY SEQUENCE#;
```

FILE_NAME	SEQ#	FIRST_CHANGE#	NEXT_CHANGE#	TIMESTAM	BEG	END	THR#
/oracle/dbs/hq_sat_2.log	2	101579	101588	11:02:57	NO	NO	1
/oracle/dbs/hq_sat_3.log	3	101588	142065	11:02:01	NO	NO	1
/oracle/dbs/hq_sat_4.log	4	142065	142307	11:02:09	NO	NO	1
/oracle/dbs/hq_sat_5.log	5	142307	142739	11:02:47	YES	YES	1
/oracle/dbs/hq_sat_6.log	6	142739	143973	12:02:09	NO	NO	1
/oracle/dbs/hq_sat_7.log	7	143973	144042	01:02:00	NO	NO	1
/oracle/dbs/hq_sat_8.log	8	144042	144051	01:02:00	NO	NO	1
/oracle/dbs/hq_sat_9.log	9	144051	144054	01:02:15	NO	NO	1
/oracle/dbs/hq_sat_10.log	10	144054	144057	01:02:20	NO	NO	1
/oracle/dbs/hq_sat_11.log	11	144057	144060	01:02:25	NO	NO	1
/oracle/dbs/hq_sat_13.log	13	144089	144147	01:02:40	NO	NO	1

Notice that for log file 11, the SCN of 144059 (recorded in Step 2) is between the FIRST_CHANGE# column value of 144057 and the NEXT_CHANGE# column value of 144060. This indicates log file 11 is currently being applied. Also, notice the gap in the sequence numbers in the SEQ# column; in the example, the gap indicates that SAT database is missing archived redo log file 12.

Step 4 Connect to the NYC database.

Issue a SQL statement such as the following:

```
SQL> CONNECT SYS/CHANGE_ON_INSTALL AS SYSDBA;
```

Step 5 Determine the highest applied SCN and highest applicable SCN on the NYC database.

Query the following columns in the V\$LOGSTDBY_PROGRESS view:

```
SQL> SELECT APPLIED_SCN, LATEST_SCN FROM V$LOGSTDBY_PROGRESS;
```

```
APPLIED_SCN LATEST_SCN
-----
143970      144146
```

Step 6 Obtain a list of the log files that were processed or are currently pending processing on the NYC database.

Issue a SQL statement such as the following:

```
SQL> SELECT SUBSTR(FILE_NAME,1,25) FILE_NAME, SUBSTR(SEQUENCE#,1,4) "SEQ#",
2> FIRST_CHANGE#, NEXT_CHANGE#, TO_CHAR(TIMESTAMP, 'HH:MI:SS') TIMESTAMP,
```

```
3> DICT_BEGIN BEG, DICT_END END, SUBSTR(THREAD#,1,4) "THR#"
4> FROM DBA_LOGSTDBY_LOG ORDER BY SEQUENCE#;
```

FILE_NAME	SEQ#	FIRST_CHANGE#	NEXT_CHANGE#	TIMESTAM	BEG	END	THR#
/oracle/dbs/hq_nyc_2.log	2	101579	101588	11:02:58	NO	NO	1
/oracle/dbs/hq_nyc_3.log	3	101588	142065	11:02:02	NO	NO	1
/oracle/dbs/hq_nyc_4.log	4	142065	142307	11:02:10	NO	NO	1
/oracle/dbs/hq_nyc_5.log	5	142307	142739	11:02:48	YES	YES	1
/oracle/dbs/hq_nyc_6.log	6	142739	143973	12:02:10	NO	NO	1
/oracle/dbs/hq_nyc_7.log	7	143973	144042	01:02:11	NO	NO	1
/oracle/dbs/hq_nyc_8.log	8	144042	144051	01:02:01	NO	NO	1
/oracle/dbs/hq_nyc_9.log	9	144051	144054	01:02:16	NO	NO	1
/oracle/dbs/hq_nyc_10.log	10	144054	144057	01:02:21	NO	NO	1
/oracle/dbs/hq_nyc_11.log	11	144057	144060	01:02:26	NO	NO	1
/oracle/dbs/hq_nyc_12.log	12	144060	144089	01:02:30	NO	NO	1
/oracle/dbs/hq_nyc_13.log	13	144089	144147	01:02:41	NO	NO	1

Notice that for log file 6, the SCN of 143970 (recorded in Step 5) is between the FIRST_CHANGE# column value of 142739 and the NEXT_CHANGE# column value of 143973. This indicates that log file 6 is currently being applied. Also, notice that there are no gaps in the sequence of log files that remain to be processed.

Step 7 Choose the best target standby database.

In most cases, the logical standby database you choose as a failover target should provide a balance between risk of data loss and time required to perform the role transition. As you analyze this information to make a decision about the best failover candidate in this scenario, consider the following:

- For minimal risk of data loss during a failover, you should choose the NYC database as the best target standby database because Steps 5 and 6 revealed that the NYC site has the most recoverable archived redo log files.
- For minimal primary database downtime during the failover, you should choose the SAT database as the best target standby database. This database is a more appropriate candidate because the queries in Steps 2 through 6 reveal that the SAT database applied 5 archived redo log files more than the NYC database (even though there was only a 1-second delay (lag) in the receipt of archived redo log files by the NYC database). However, if it is not possible to obtain and apply a copy of the missing archived redo log file (log file 12 in the example), then you will not be able to make the SAT database as current as you can the NYC database. Therefore, you will lose the unrecovered data (log files 12, 13, and part of log file 14 in the example).

Based on your business requirements, choose the best target standby database.

Step 8 Bring the selected standby database to its most current state.

If you chose the SAT database as the best target based on your business requirements, perform the following steps:

1. Manually retrieve any missing archived redo log files using an operating system utility. (This example uses the UNIX cp command.) In this case, the SAT database is missing archived redo log file 12. Because the NYC database received this archived redo log file, you can copy it from the NYC database to the SAT database, as follows:

```
%cp /net/nyc/oracle/dbs/hq_nyc_12.log
/net/sat/oracle/dbs/hq_sat_12.log
```

2. Determine if a partial archived redo log file exists for the next sequence number. In this example, the next sequence number should be 14. The following UNIX command shows the directory on the SAT database, looking for the presence of an archived redo log file named hq_sat_14.log:

```
%ls -l /net/sat/oracle/dbs/hq_sat_14.log
-rw-rw---- 1 oracle  dbs 333280 Feb 12 1:03 hq_sat_14.log
```

3. Stop log apply services and register both the retrieved archived redo log file and the partial archived redo log file:

```
SQL> ALTER DATABASE STOP LOGICAL STANDBY APPLY;
SQL> ALTER DATABASE REGISTER LOGICAL LOGFILE '/oracle/dbs/hq_sat_12.log';
SQL> ALTER DATABASE REGISTER LOGICAL LOGFILE '/oracle/dbs/hq_sat_14.log';
```

4. Start log apply services to apply the most current log file:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY;
```

5. Determine the highest applied SCN on the SAT database by querying the V\$LOGSTDBY_PROGRESS view to see if the value of the APPLIED_SCN column is equal to the value of the LATEST_SCN column:

```
SQL> SELECT APPLIED_SCN, LATEST_SCN FROM V$LOGSTDBY_PROGRESS;
```

```
APPLIED_SCN LATEST_SCN
-----
144205      144205
```

Because the SCN values match, you can be assured that there is no longer a delay (lag) between the current log file on the primary database and the last log file applied to the SAT database.

If you chose the NYC database as the best target based on your business requirements, perform the following steps:

1. Determine if a partial archived redo log file exists for the next sequence number. In this example, the next sequence number should be 14. The following UNIX command shows the directory on the NYC database, looking for the presence of an archived redo log file named hq_nyc_14:

```
%ls -l /net/nyc/oracle/dbs/hq_nyc_14.log
-rw-rw---- 1 oracle  dbs 333330 Feb 12 1:03 hq_nyc_14.log
```

2. Register the partial archived redo log file on the NYC database:

```
SQL> ALTER DATABASE STOP LOGICAL STANDBY APPLY;
SQL> ALTER DATABASE REGISTER LOGICAL LOGFILE '/oracle/dbs/hq_nyc_14.log';
```

3. Start log apply services to apply the most current log file:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY;
```

4. Determine the highest applied SCN on the NYC database by querying the V\$LOGSTDBY_PROGRESS view to see if the value of the APPLIED_SCN column is equal to the value of the LATEST_SCN column:

```
SQL> SELECT APPLIED_SCN, LATEST_SCN FROM V$LOGSTDBY_PROGRESS;
```

```
APPLIED_SCN LATEST_SCN
-----
144205      144205
```

Because the SCN values match, you can be sure there is no longer a delay (lag) between the current log file on the primary database and the last log file received and applied by the NYC database.

Step 9 Perform the failover.

You are now ready to stop log apply services and fail over the selected logical standby database to the primary role.

See [Section 7.3.2](#) for additional information on how to perform the failover.

12.3 Configuring a Logical Standby Database to Support a New Primary Database

This section presents the steps required on a logical standby database after the primary database has failed over to another standby database. After a failover has occurred, a logical standby database cannot act as a standby database for the new primary database until it has applied the final redo from the original primary database. This is similar to the way the new primary database applied the final redo during the failover. The steps you must perform depend on whether the new primary database was a physical standby or a logical standby database prior to the failover:

- [Section 12.3.1, "When the New Primary Database Was Formerly a Physical Standby Database"](#)
- [Section 12.3.2, "When the New Primary Database Was Formerly a Logical Standby Database"](#)

12.3.1 When the New Primary Database Was Formerly a Physical Standby Database

This scenario demonstrates how to configure the SAT logical standby database with the failed over NYC database. This scenario is a continuation of the example described in [Section 12.2.2](#). However, in this example, the NYC database was formerly a physical standby database.

Perform the following steps to configure the logical standby database with the new primary database:

Step 1 Disable archiving from the primary database.

On the NYC database, issue the following statements (assuming LOG_ARCHIVE_DEST_4 is configured to archive to the SAT database):

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_4=DEFER;  
SQL> ALTER SYSTEM ARCHIVE LOG CURRENT;
```

Step 2 Verify the logical standby database is capable of serving as a standby database to the new primary database.

On the SAT database, issue the following statement:

```
SQL> EXECUTE DBMS_LOGSTDBY.PREPARE_FOR_NEW_PRIMARY (-  
    former_standby_type => 'PHYSICAL' -  
    dblink => 'nyc_link');
```

Note: If the ORA-16109 message is returned and the 'LOGSTDBY: prepare_for_new_primary failure -- applied too far, flashback required.' warning is written in the alert.log, perform the following steps:

1. Flash back the database to the SCN as stated in the warning and then
2. Repeat this step before continuing.

See [Section 12.4.3](#) for an example of how to flash back a logical standby database to an Apply SCN.

Step 3 Enable archiving on the primary database.

On the NYC database, issue the following statements (assume LOG_ARCHIVE_DEST_4 is configured to archive to the SAT database):

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_4=ENABLE;
SQL> ALTER SYSTEM ARCHIVE LOG CURRENT;
```

Step 4 Query the new primary database to determine the SCN at which real-time apply can be enabled on the logical standby database

On the NYC database, issue the following query to determine the SCN of interest:

```
SQL> SELECT MAX(NEXT_CHANGE#) -1 AS WAIT_FOR_SCN FROM V$ARCHIVED_LOG;
```

Step 5 Start SQL Apply.

On the SAT database, issue the following statement:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY;
```

Note that you must always issue this statement without the real-time apply option. You need to wait for SQL Apply to apply past WAIT_FOR_SCN returned in Step 4, before you can enable real-time apply. To determine when it is safe to resume real-time apply on the logical standby database, monitor the V\$LOGSTDBY_PROGRESS view:

```
SQL> SELECT APPLIED_SCN FROM V$LOGSTDBY_PROGRESS;
```

When the value returned is greater than or equal to the WAIT_FOR_SCN value returned in Step 4, you can stop SQL Apply and restart it with real-time apply option:

```
SQL> ALTER DATABASE STOP LOGICAL STANDBY APPLY;
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE;
```

12.3.2 When the New Primary Database Was Formerly a Logical Standby Database

This scenario demonstrates how to configure the SAT logical standby database with the failed over NYC database. This scenario is a continuation of the example described in [Section 12.2.2](#). However, in this example, the NYC database was formerly a logical standby database.

Perform the following steps to configure the logical standby database with the new primary database:

Step 1 Ensure the new primary database is ready to support logical standby databases.

On the NYC database, ensure the following query returns a value of READY. Otherwise, the LSP1 background process has not completed its work and the configuration of this logical must wait. For example:

```
SQL> SELECT VALUE FROM DBA_LOGSTDBY_PARAMETERS WHERE
2> NAME = 'REINSTATEMENT_STATUS';
```

```
VALUE
-----
READY
```

Note: If the VALUE column contains NOT POSSIBLE it means that no logical standby database may be configured with the new primary database, and you must reinstate the database.

Step 2 Disable archiving from the primary database.

On the NYC database, issue the following statements (assume LOG_ARCHIVE_DEST_4 is configured to archive to the SAT database):

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_4=DEFER;
SQL> ALTER SYSTEM ARCHIVE LOG CURRENT;
```

Step 3 Verify the logical standby database is capable of being a standby to the new primary.

On the SAT database, issue the following statement:

```
SQL> EXECUTE DBMS_LOGSTDBY.PREPARE_FOR_NEW_PRIMARY (-
former_standby_type => 'LOGICAL' -
dblink => 'nyc_link');
```

Note: If the ORA-16109 message is returned and the 'LOGSTDBY: prepare_for_new_primary failure -- applied too far, flashback required.' warning is written in the alert.log file, perform the following steps:

1. Flash back the database to the SCN as stated in the warning and then
2. Repeat this step before continuing.

See [Section 12.4.3](#) for an example of how to flash back a logical standby database to an Apply SCN.

Step 4 Determine the log files that must be copied to the local system.

On the SAT database, look for the output from the DBMS_LOGSTDBY.PREPARE_FOR_NEW_PRIMARY procedure that identifies the log files that must be copied to the local system. If Step 3 identified the failover as a *no-data-loss* failover, then the displayed log files must be copied from the new primary database and should not be obtained from other logical standby databases or the former primary database. For example, on a Linux system, you would enter the `grep` command:

```
%grep 'LOGSTDBY: Terminal log' alert_sat.log
LOGSTDBY: Terminal log: [/oracle/dbs/hq_nyc_13.log]
```

Note: If the prior step was executed multiple times, the output from the most recent attempt is the only relevant output. File paths are relative to the new primary database and may not be resolvable on the local file system.

Step 5 Copy the log files to the local system.

On the SAT database, copy the terminal log files to the local system. The following example shows how to do this using Linux commands:

```
%cp /net/nyc/oracle/dbs/hq_nyc_13.log
/net/sat/oracle/dbs/hq_sat_13.log
```

Step 6 Register the terminal log with logical standby database.

On the SAT database, issue the following statement:

```
SQL> ALTER DATABASE REGISTER OR REPLACE LOGICAL LOGFILE -
'/net/sat/oracle/dbs/hq_sat_13.log';
```

Step 7 Start SQL Apply.

On the SAT database, issue the following statements:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY NEW PRIMARY nyc_link;
```

Note that you must always issue this statement without the real-time apply option. If you want to enable real-time apply on the logical standby database, wait for the above statement to complete successfully, and then issue the following statements:

```
SQL> ALTER DATABASE STOP LOGICAL STANDBY APPLY;
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE;
```

Step 8 Enable archiving on the primary database to the logical standby database.

On the NYC database, issue the following statements (assuming LOG_ARCHIVE_DEST_4 is configured to archive to the SAT database):

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_4=ENABLE;
SQL> ALTER SYSTEM ARCHIVE LOG CURRENT;
```

12.4 Using Flashback Database After a Failover

After a failover occurs, the original primary database can no longer participate in the Data Guard configuration until it is repaired and established as a standby database in the new configuration. To do this, you can use the Flashback Database feature to recover the failed primary database to a point in time before the failover occurred, and then convert it into a physical or logical standby database in the new configuration. The following sections describe:

- [Flashing Back a Failed Primary Database into a Physical Standby Database](#)
- [Flashing Back a Failed Primary Database into a Logical Standby Database](#)

Note: You must have already enabled Flashback Database on the original primary database before the failover. See *Oracle Database Backup and Recovery Basics* for more information.

- [Flashing Back a Logical Standby Database to a Specific Applied SCN](#)

See Also: Oracle Data Guard Broker for automatic reinstatement of the failed primary database as a new standby database (as an alternative to using Flashback Database)

12.4.1 Flashing Back a Failed Primary Database into a Physical Standby Database

The following steps assume the user has already performed a failover involving a physical standby database and Flashback Database has been enabled on the old primary database. This procedure brings the old primary database back into the Data Guard configuration as a new physical standby database.

Step 1 Determine the SCN at which the old standby database became the primary database.

On the new primary database, issue the following query to determine the SCN at which the old standby database became the new primary database:

```
SQL> SELECT TO_CHAR(STANDBY_BECAME_PRIMARY_SCN) FROM V$DATABASE;
```

Step 2 Flash back the failed primary database.

To create a new physical standby database, shut down the old primary database (if necessary), mount it, and flash it back to the value for `STANDBY_BECAME_PRIMARY_SCN` that was determined in Step 1:

```
SQL> SHUTDOWN IMMEDIATE;
SQL> STARTUP MOUNT;
SQL> FLASHBACK DATABASE TO SCN standby_became_primary_scn;
```

Step 3 Convert the database to a physical standby database.

Perform the following steps on the old primary database:

1. Issue the following statement on the old primary database:

```
SQL> ALTER DATABASE CONVERT TO PHYSICAL STANDBY;
```

This statement will dismount the database after successfully converting the control file to a standby control file.

2. Shut down and restart the database:

```
SQL> SHUTDOWN IMMEDIATE;
SQL> STARTUP MOUNT;
```

Step 4 Restart transporting redo to the new physical standby database.

Before the new standby database was created, the new primary database probably stopped transmitting redo to the remote destination. To restart redo transport services, perform the following steps on the new primary database:

1. Issue the following query to see the current state of the archive destinations:

```
SQL> SELECT DEST_ID, DEST_NAME, STATUS, PROTECTION_MODE, DESTINATION, ERROR, SRL
2> FROM V$ARCHIVE_DEST_STATUS;
```

2. If necessary, enable the destination:

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_n=ENABLE;
```

3. Perform a log switch to ensure the standby database begins receiving redo data from the new primary database, and verify it was sent successfully. At the SQL prompt, enter the following statements:

```
SQL> ALTER SYSTEM SWITCH LOGFILE;
SQL> SELECT DEST_ID, DEST_NAME, STATUS, PROTECTION_MODE, DESTINATION, ERROR,SRL
2> FROM V$ARCHIVE_DEST_STATUS;
```

On the new standby database, you may also need to change the LOG_ARCHIVE_DEST_1 initialization parameters so that redo transport services do not transmit redo data to other databases. This step can be skipped if both the primary and standby database roles were set up with the VALID_FOR attribute in one server parameter file (SPFILE). By doing this, the Data Guard configuration operates properly after a role transition.

Step 5 Start Redo Apply.

Start Redo Apply or real-time apply on the new physical standby database:

- To start Redo Apply:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE DISCONNECT;
```

- To start real-time apply:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE
2> USING CURRENT LOGFILE DISCONNECT;
```

Once the failed primary database is restored and is running in the standby role, you can optionally perform a switchover to transition the databases to their original (pre-failure) roles. See [Section 7.2.1, "Switchovers Involving a Physical Standby Database"](#) for more information.

12.4.2 Flashing Back a Failed Primary Database into a Logical Standby Database

The following steps assume that the Data Guard configuration has already completed a failover involving a logical standby database and Flashback Database has been enabled on the old primary database. This procedure brings the old primary database back into the Data Guard configuration as a new logical standby database, without having to formally reinitialize it from the new primary database.

Step 1 Determine the SCN to which to flash back the failed primary database.

On the new primary database, issue the following query to determine the SCN to which you want to flash back the failed primary database:

```
SQL> SELECT APPLIED_SCN AS FLASHBACK_SCN FROM V$LOGSTDBY_PROGRESS;
```

Step 2 Determine the log files that must be copied to the failed primary database for Flashback Database.

On the new primary database, issue the following query to determine the log files that must be copied to the failed primary database for Flashback Database to reach a consistent state

```
SQL> SELECT NAME FROM DBA_LOGSDBY_LOG
2> WHERE NEXT_CHANGE# >
3>         (SELECT VALUE FROM DBA_LOGSTDBY_PARAMETERS
4> WHERE NAME = 'STANDBY_BECAME_PRIMARY_SCN')
5> AND FIRST_CHANGE <= (FLASHBACK_SCN from step 1);
```

Step 3 Flash back the failed primary database.

To create a new logical standby database, shut down the database (if necessary), mount the failed primary database, flash it back to the FLASHBACK_SCN determined in step 1, and enable the database guard.

```
SQL> SHUTDOWN IMMEDIATE;
SQL> STARTUP MOUNT;
SQL> FLASHBACK DATABASE TO SCN became_primary_scn;
SQL> ALTER DATABASE GUARD ALL;
```

Step 4 Open the database with the RESETLOGS option.

```
SQL> ALTER DATABASE OPEN RESETLOGS;
```

Step 5 Create a database link to the new primary database and start SQL Apply.

```
SQL> CREATE PUBLIC DATABASE LINK mylink
  2> CONNECT TO system IDENTIFIED BY password
  3> USING 'service_name_of_new_primary_database';

SQL> ALTER DATABASE START LOGICAL STANDBY APPLY NEW PRIMARY mylink;
```

The role reversal is now complete.

Once the failed primary database has been restored and is running in the standby role, you can optionally perform a switchover to transition the databases to their original (pre-failure) roles. See [Section 7.3.1, "Switchovers Involving a Logical Standby Database"](#) for more information.

12.4.3 Flashing Back a Logical Standby Database to a Specific Applied SCN

One of the benefits of a standby database is that Flashback Database can be performed on the standby database without affecting the primary database service. Flashing back a database to a specific point in time is a straightforward task, however on a logical standby database, you may want to flash back to a time just before a known transaction was committed. Such a need can arise when configuring a logical standby database with a new primary database after a failover.

The following steps describe how to use Flashback Database and SQL Apply to recover to a known applied SCN.

Step 1 Identify the log files that contain the Apply SCN.

On the logical standby database, issue the following query to identify the log files that contain the Apply_SCN:

```
SQL> SELECT FILE_NAME FROM DBA_LOGSTDBY_LOG
  5> WHERE FIRST_CHANGE# <= APPLY_SCN
  6> AND NEXT_CHANGE# > APPLY_SCN
  7> ORDER BY FIRST_CHANGE# ASCENDING;
```

```
FILE_NAME
```

```
-----
/net/sat/oracle/dbs/hq_sat_13.log
```

Step 2 Locate the timestamp associated with the SQL Apply initial reading of the first log file.

Locate the timestamp in the alert.log file associated with the SQL Apply initial reading of the first log file displayed in Step 1. For example:

```
%grep -B 1 '^LOGMINER: Begin mining logfile' alert_gap2.log | grep -B 1 hq_sat_
```

13.log

```
Tue Jun 7 02:38:18 2005
LOGMINER: Begin mining logfile: /net/sat/oracle/dbs/hq_sat_13.log
```

Step 3 Flash back the database to the timestamp.

Flash back the database to the timestamp identified in Step 2.

```
SQL> SHUTDOWN;
SQL> STARTUP MOUNT EXCLUSIVE;
SQL> FLASHBACK DATABASE TO TIMESTAMP -
      TO_TIMESTAMP('07-Jun-05 02:38:18', 'DD-Mon-RR HH24:MI:SS');
SQL> ALTER DATABASE OPEN RESETLOGS;
```

Step 4 Confirm SQL Apply has applied less than or up to the APPLY_SCN

Issue the following query:

```
SQL> SELECT APPLIED_SCN FROM V$LOGSTDBY_PROGRESS;
```

12.5 Using Flashback Database After Issuing an Open Resetlogs Statement

Suppose an error has occurred on the primary database in a Data Guard configuration in which the standby database is using real-time apply. In this situation, the same error will be applied on the standby database.

However, if Flashback Database is enabled, you can revert the primary and standby databases back to their pre-error condition by issuing the `FLASHBACK DATABASE` and `OPEN RESETLOGS` statements on the primary database, and then issuing a similar `FLASHBACK STANDBY DATABASE` statement on the standby database before restarting log apply services. (If Flashback Database is not enabled, you need to re-create the standby database, as described in [Chapter 3](#) and [Chapter 4](#), after the point-in-time recovery was performed on the primary database.)

12.5.1 Flashing Back a Physical Standby Database to a Specific Point-in-Time

The following steps describe how to avoid re-creating a physical standby database after you issued the `OPEN RESETLOGS` statement on the primary database.

Step 1 Determine the SCN before the RESETLOGS operation occurred.

On the primary database, use the following query to obtain the value of the system change number (SCN) that is 2 SCNs before the `RESETLOGS` operation occurred on the primary database:

```
SQL> SELECT TO_CHAR(RESETLOGS_CHANGE# - 2) FROM V$DATABASE;
```

Step 2 Obtain the current SCN on the standby database.

On the standby database, obtain the current SCN with the following query:

```
SQL> SELECT TO_CHAR(CURRENT_SCN) FROM V$DATABASE;
```

Step 3 Determine if it is necessary to flash back the database.

If the value of `CURRENT_SCN` is larger than the value of `resetlogs_change# - 2`, issue the following statement to flash back the standby database.

```
SQL> FLASHBACK STANDBY DATABASE TO SCN resetlogs_change# -2;
```

- If the value of `CURRENT_SCN` is less than the value of the `resetlogs_change# - 2`, skip to Step 4.
- If the standby database's SCN is far enough behind the primary database's SCN, log apply services will be able to continue through the `OPEN RESETLOGS` statement without stopping. In this case, flashing back the database is unnecessary because log apply services do not stop upon reaching the `OPEN RESETLOGS` statement in the redo data.

Step 4 Restart Redo Apply.

To start Redo Apply on the physical standby database, issue the following statement:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE DISCONNECT;
```

The standby database is now ready to receive and apply redo from the primary database.

12.5.2 Flash Back a Logical Standby Database After Flashing Back the Primary

The following steps describe how to avoid re-creating a logical standby database after you have flashed back the primary database and opened it by issuing `OPEN RESETLOGS` statement.

Step 1 Determine the SCN at the primary database.

On the primary database, use the following query to obtain the value of the system change number (SCN) that is 2 SCNs before the `RESETLOGS` operation occurred on the primary database:

```
SQL> SELECT TO_CHAR(RESETLOGS_CHANGE# - 2) AS FLASHBACK_SCN FROM V$DATABASE;
```

Step 2 Stop SQL Apply.

On the logical standby database, stop SQL Apply:

```
SQL> ALTER DATABASE STOP LOGICAL STANDBY APPLY;
SQL> SELECT APPLIED_SCN FROM V$LOGSTDBY_PROGRESS;
```

If the `APPLIED_SCN` is less than the value of the `resetlogs_change#-2`, you do not need to flash back the standby database and can proceed to Step 6. This may happen if SQL Apply is running with a delay. Otherwise, continue with Step 5.

Step 3 Determine which archived redo log file contains the FLASHBACK_SCN.

On the logical standby database, determine which archived redo log file contains the `FLASHBACK_SCN` determined in Step 1

```
SQL> SELECT FILE_NAME FROM DBA_LOGSTDBY_LOG
2> WHERE FIRST_CHANGE# <= FLASHBACK_SCN
3> AND NEXT_CHANGE# > FLASHBACK_SCN
4> ORDER BY FIRST_CHANGE# ASCENDING;
```

```
FILE_NAME
```

```
-----
/net/sat/oracle/dbs/hq_sat_146.log
```

Step 4 Locate the timestamp in the alert.log file.

Locate the timestamp in the `alert.log` file associated with the SQL Apply initial reading of the first log file displayed in Step 1. For example:

```
%grep -B 1 '^LOGMINER: Begin mining logfile' alert.log | grep -B 1 hq_sat_146.log
```

```
Tue Mar  7 12:38:18 2005
LOGMINER: Begin mining logfile: /net/sat/oracle/dbs/hq_sat_146.log
```

Step 5 Flash back the logical standby database to the timestamp.

Issue the following SQL statements to flash back the logical standby database to the time identified in step 4, and open the logical standby database with the `RESETLOGS` option:

```
SQL> SHUTDOWN;
SQL> STARTUP MOUNT EXCLUSIVE;
SQL> FLASHBACK DATABASE TO TIMESTAMP('07-Mar-05 12:38:18', 'DD-Mon-RR
HH24:MI:SS');
SQL> ALTER DATABASE OPEN RESETLOGS;
```

Step 6 Confirm SQL Apply has applied less than or up to the Apply SCN.

On the logical standby database, issue the following query:

```
SQL> SELECT APPLIED_SCN FROM V$LOGSTDBY_PROGRESS;
```

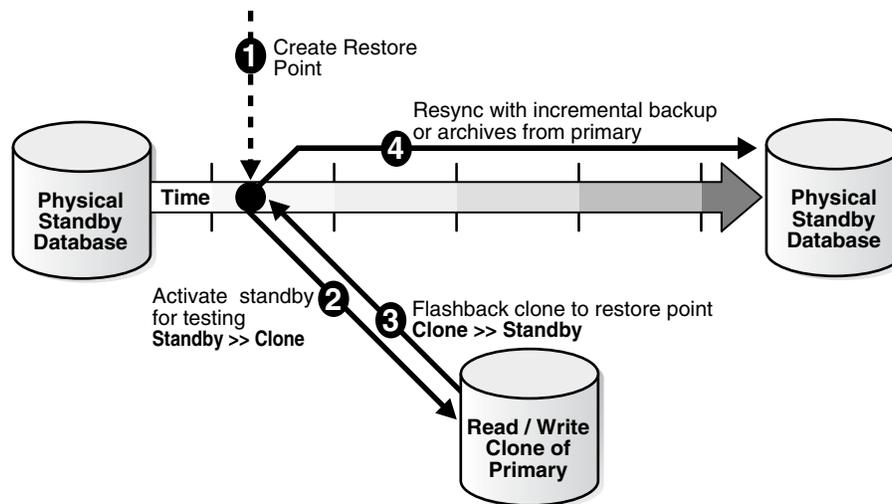
Step 7 Start SQL Apply.

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE;
```

12.6 Using a Physical Standby Database for Read/Write Testing and Reporting

Using a combination of Data Guard, restore points, and Flashback Database, a physical standby database can be opened temporarily in read/write mode for development, reporting, or testing purposes, and then flashed back to a point in the past to be reverted back to a physical standby database. When the database is flashed back, Data Guard automatically synchronizes the standby database with the primary database, without the need to re-create the physical standby database from a backup copy of the primary database.

[Figure 12-7](#) shows a physical standby database being activated as a read/write clone database, resynchronized with the primary database, and eventually flashed back and reverted to its physical standby database role. You can repeat this cycle of activate, flashback and revert as many times as is necessary.

Figure 12–7 Using a Physical Standby Database As a Testing and Reporting Database

Caution: While the database is activated, it is not receiving redo data from the primary database and cannot provide disaster protection. It is recommended that there be at least two physical standby databases participating in the configuration so that the primary database remains protected against data loss.

Perform the following steps to activate the physical standby database as a production database and later resynchronize it with the primary database.

Step 1 Prepare the physical standby database to be activated.

1. Set up a flash recovery area.

On the physical standby database that will be activated for read/write access, you should set the following initialization parameters to ensure a guaranteed restore point can be created. This scenario sets up the flash recovery area in the /arch/oradata location:

```
SQL> ALTER SYSTEM SET DB_RECOVERY_FILE_DEST_SIZE=20G;
SQL> ALTER SYSTEM SET DB_RECOVERY_FILE_DEST='/arch/oradata';
```

2. Cancel Redo Apply and create a guaranteed restore point.

On the physical standby database, stop Redo Apply and create a restore point:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE CANCEL;
SQL> CREATE RESTORE POINT before_application_patch GUARANTEE FLASHBACK
DATABASE;
```

When you create a guaranteed restore point, you associate an easy-to-remember name with a timestamp or SCN so that you can later flash back the database to a name instead of specifying an exact SCN or time.

Step 2 Prepare the primary database to have the physical standby be diverged.

1. Archive the current log file.

On the primary database, switch logs so the SCN of the restore point (created in step 1) will be archived on the physical standby database:

```
SQL> ALTER SYSTEM ARCHIVE LOG CURRENT;
```

When using standby redo log files, this step is essential to ensure the database can be properly flashed back to the restore point.

2. Defer log archive destinations pointing to the standby that will be activated.

On the primary database (on all instances if this is a Real Applications Cluster), defer the archival of redo data to the destination associated with the physical standby database that will be opened. For example:

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2=DEFER;
```

Step 3 Activate the physical standby database.

On the physical standby database, perform the following steps:

1. Activate the physical standby database:

```
SQL> ALTER DATABASE ACTIVATE STANDBY DATABASE;
```

2. If the physical standby database has been opened read-only since the instance was started, perform this step. Otherwise, skip to step 3.

Enter the following statement to shut down and restart the physical standby database:

```
SQL> SHUTDOWN STANDBY FORCE;
```

3. Set the protection mode to maximum performance and open the database for read/write access:

```
SQL> ALTER DATABASE SET STANDBY DATABASE TO MAXIMIZE PERFORMANCE;
SQL> ALTER DATABASE OPEN;
```

After the standby database is activated, its protection mode is downgraded to the maximum performance mode, because there is no standby database configured to protect the database against data loss while it is temporarily activated as a production database. Note that this protection mode setting does not affect the protection mode of the original primary database, it affects only the activated standby database.

When the activated standby database is converted back to a physical standby database, its protection mode is automatically changed to match that of the original primary database.

If the standby database that was opened read/write temporarily has remote archive log destinations, you might need to disable them. In this way, the read/write testing or reporting database will not propagate its temporary changes to other standby databases in the original Data Guard environment.

Step 4 Use the activated database for reporting or testing.

Once the standby database has been activated, you can run reporting tools or perform other testing and activities for days or even weeks, independent of the primary database.

Caution: While the database is activated, it is not receiving redo data from the primary database and cannot provide disaster protection. It is recommended that there be at least two physical standby databases participating in the configuration so that the primary database remains protected against data loss.

Also, any results stored in the activated database will be lost when you later flash back the database. Results that should be saved must be copied out of the activated database before flashing it back.

Step 5 Revert the activated database back to a physical standby database.

After you finish testing, you need to resynchronize the activated database with the primary database. Issue the following statements on the activated database to quickly flash it back to the guaranteed restore point and resynchronize it with the primary database:

```
SQL> STARTUP MOUNT FORCE;
SQL> FLASHBACK DATABASE TO RESTORE POINT before_application_patch;
SQL> ALTER DATABASE CONVERT TO PHYSICAL STANDBY;
SQL> STARTUP MOUNT FORCE;
```

Step 6 Catch up the standby database to the primary database.

The method you use will depend on how far the activated standby database lags behind the primary database in its application of redo data:

- Let archive gap resolution fetch all missing archived redo log files and allow Redo Apply to apply the gap.

If the activated database has not fallen too far behind the original primary database, issue the following statement on the standby database to resynchronize it with the primary database and restart Redo Apply. For example:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE DISCONNECT;
```

Then, go to Step 7.

- Create an incremental backup on the primary and apply it to the standby.

If the activated database has fallen too far behind the original primary database (for example, if there are not sufficient log files available), you can take an incremental backup from the primary database and apply it to the standby database. See [Section 12.7.1](#) for information about using RMAN incremental backups to resynchronize the standby database with the primary database.

Note: If the standby database lags very far behind the primary database, it may be quicker to apply an incremental backup created from the primary database using the steps in [Section 12.7.1](#).

After you apply an incremental backup to the standby database, you typically need to apply more redo to the standby database to activate the physical standby database again for read/write testing or reporting purposes. More specifically, you might need to apply the redo generated by the primary database while the incremental backup was taken. Otherwise, issuing an `ALTER DATABASE ACTIVATE STANDBY DATABASE` will return an error.

Step 7 Reenable archiving to the physical standby database destination.

On the primary database, issue the following statement to reenable archiving to the physical standby database:

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_2=ENABLE;
```

12.7 Using RMAN Incremental Backups to Roll Forward a Physical Standby Database

In some situations, RMAN incremental backups can be used to synchronize a physical standby database with the primary database. Using the `RMAN BACKUP INCREMENTAL FROM SCN` command, you can create a backup on the primary database that starts at the standby database's current SCN, which can then be used to roll the standby database forward in time.

The following sections describe situations in which RMAN incremental backups may be useful:

- [Physical Standby Database Lags Far Behind the Primary Database](#)
- [Physical Standby Database Has Nologging Changes On a Subset of Datafiles](#)
- [Physical Standby Database Has Widespread Nologging Changes](#)

See Also: *Oracle Database Backup and Recovery Advanced User's Guide* for more information about RMAN incremental backups

12.7.1 Physical Standby Database Lags Far Behind the Primary Database

Follow this step-by-step procedure to roll forward a physical standby database that has fallen far behind the primary database.

Note: The steps in this section can also be used to resolve problems if a physical standby database has lost or corrupted archived redo data or has an unresolvable archive gap.

1. On the standby database, query the `V$DATABASE` view and record the current SCN of the standby database:

```
SQL> SELECT CURRENT_SCN FROM V$DATABASE;
CURRENT_SCN
-----
      233995
```

2. Stop Redo Apply on the standby database:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE CANCEL;
```

3. Connect to the primary database as the RMAN target and create an incremental backup from the current SCN of the standby database that was recorded in step 1:

```
RMAN> BACKUP INCREMENTAL FROM SCN 233995 DATABASE FORMAT '/tmp/ForStandby_%U'
tag 'FOR STANDBY';
```

Note: RMAN does not consider the incremental backup as part of a backup strategy at the source database. Hence:

- The backup is not suitable for use in a normal RECOVER DATABASE operation at the source database
 - The backup is not cataloged at the source database
 - The backup sets produced by this command are written to the /dbs location by default, even if the flash recovery area or some other backup destination is defined as the default for disk backups.
 - You must create this incremental backup on disk for it to be useful. When you move the incremental backup to the standby database, you must catalog it at the standby as described in *Oracle Database Backup and Recovery Advanced User's Guide*. Backups on tape cannot be cataloged.
-
-

4. Transfer all backup sets created on the primary system to the standby system (note that there may be more than one backup file created):

```
SCP /tmp/ForStandby_* standby:/tmp
```

5. Connect to the standby database as the RMAN target, and catalog all incremental backup pieces:

```
RMAN> CATALOG START WITH '/tmp/ForStandby_*';
```

6. Connect to the standby database as the RMAN target and apply incremental backups

```
RMAN> RECOVER DATABASE NOREDO;
```

7. Remove the incremental backups from the standby system:

```
RMAN> DELETE BACKUP TAG 'FOR STANDBY';
```

8. Manually remove the incremental backups from the primary system:

```
rm /tmp/ForStandby_*
```

9. Start Redo Apply on the physical standby database:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE USING CURRENT LOGFILE  
DISCONNECT;
```

12.7.2 Physical Standby Database Has Nologging Changes On a Subset of Datafiles

Follow this step-by-step procedure to roll forward a physical standby database for which nologging changes have been applied to a small subset of the database:

1. List the files that have had nologging changes applied by querying the V\$DATAFILE view on the standby database. For example:

```
SQL> SELECT FILE#, FIRST_NONLOGGED_SCN FROM V$DATAFILE  
2> WHERE FIRST_NONLOGGED_SCN > 0;
```

```
FILE#          FIRST_NONLOGGED_SCN  
-----  
         4                225979
```

2. Stop Redo Apply on the standby database:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE CANCEL;
```

3. On the standby database, offline the datafiles (recorded in step 1) that have had nologging changes. Taking these datafiles offline ensures redo data is not skipped for the corrupt blocks while the incremental backups are performed.

```
SQL> ALTER DATABASE DATAFILE 4 OFFLINE FOR DROP;  
SQL> ALTER DATABASE DATAFILE 5 OFFLINE FOR DROP;
```

4. Start Redo Apply on the standby database:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE  
2> USING CURRENT LOGFILE DISCONNECT;
```

5. While connected to the primary database as the RMAN target, create an incremental backup for each datafile listed in the FIRST_NONLOGGED_SCN column (recorded in step 1). For example:

```
RMAN> BACKUP INCREMENTAL FROM SCN 225979 DATAFILE 4 FORMAT '/tmp/ForStandby_%U'  
TAG 'FOR STANDBY';  
RMAN> BACKUP INCREMENTAL FROM SCN 230184 DATAFILE 5 FORMAT '/tmp/ForStandby_%U'  
TAG 'FOR STANDBY';
```

6. Transfer all backup sets created on the primary system to the standby system. (Note that there may be more than one backup file created.)

```
SCP /tmp/ForStandby_* standby:/tmp
```

7. While connected to the physical standby database as the RMAN target, catalog all incremental backup pieces. For example:

```
RMAN> CATALOG START WITH '/tmp/ForStandby_';
```

8. Stop Redo Apply on the standby database:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE CANCEL;
```

9. Online the datafiles on the standby database

```
SQL> ALTER DATABASE DATAFILE 4 ONLINE;  
SQL> ALTER DATABASE DATAFILE 5 ONLINE;
```

10. While connected to the physical standby database as the RMAN target, apply the incremental backup sets:

```
RMAN> RECOVER DATAFILE 4, 5 NOREDO;
```

11. Query the V\$DATAFILE view on the standby database to verify there are no datafiles with nologged changes. The following query should return zero rows

```
SQL> SELECT FILE#, FIRST_NONLOGGED_SCN FROM V$DATAFILE  
2> WHERE FIRST_NONLOGGED_SCN > 0;
```

12. Remove the incremental backups from the standby system:

```
RMAN> DELETE BACKUP TAG 'FOR STANDBY';
```

13. Manually remove the incremental backups from the primary system. For example, the following example uses the Linux rm command:

```
rm /tmp/ForStandby_*
```

14. Start Redo Apply on the standby database:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE USING CURRENT LOGFILE
DISCONNECT;
```

12.7.3 Physical Standby Database Has Widespread Nologging Changes

Follow this step-by-step procedure to roll forward a physical standby database for which nologging changes have been applied to a large portion of the database:

1. Query the V\$DATAFILE view on the standby database to record the lowest FIRST_NONLOGGED_SCN:

```
SQL> SELECT MIN(FIRST_NONLOGGED_SCN) FROM V$DATAFILE
2> WHERE FIRST_NONLOGGED_SCN>0;
```

```
MIN(FIRST_NONLOGGED_SCN)
-----
223948
```

2. Stop Redo Apply on the standby database:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE CANCEL;
```

3. While connected to the primary database as the RMAN target, create an incremental backup from the lowest FIRST_NONLOGGED_SCN (recorded in step 1)

```
RMAN> BACKUP INCREMENTAL FROM SCN 223948 DATABASE FORMAT '/tmp/ForStandby_%U'
tag 'FOR STANDBY';
```

4. Transfer all backup sets created on the primary system to the standby system. (Note that more than one backup file may have been created.) The following example uses the `scp` command to copy the files:

```
scp /tmp/ForStandby_* standby:/tmp
```

5. While connected to the standby database as the RMAN target, catalog all incremental backup piece(s)

```
RMAN> CATALOG START WITH '/tmp/ForStandby_*';
```

6. While connected to the standby database as the RMAN target, apply the incremental backups:

```
RMAN> RECOVER DATABASE NOREDO;
```

7. Query the V\$DATAFILE view to verify there are no datafiles with nologged changes. The following query on the standby database should return zero rows:

```
SQL> SELECT FILE#, FIRST_NONLOGGED_SCN FROM V$DATAFILE
2> WHERE FIRST_NONLOGGED_SCN > 0;
```

8. Remove the incremental backups from the standby system:

```
RMAN> DELETE BACKUP TAG 'FOR STANDBY';
```

9. Manually remove the incremental backups from the primary system. For example, the following removes the backups using the Linux `rm` command:

```
rm /tmp/ForStandby_*
```

10. Start Redo Apply on the standby database:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE
2> USING CURRENT LOGFILE DISCONNECT;
```

12.8 Using a Physical Standby Database with a Time Lag

By default, when log apply services are running on the standby database, the redo data is either written to archived log files and applied, or when real-time apply is enabled, the redo is written to the standby database as it arrives from the primary database. But in some cases, you may want to create a time lag between the archiving of an online redo log file at the primary site and the application of the archived redo log file at the standby site. A time lag can protect against the transfer of corrupted or erroneous data from the primary site to the standby site. When you set a time delay, it does not delay the transport of the redo data to the standby database. Instead, the time lag you specify begins when the redo data is completely archived at the standby destination.

For example, suppose you run a batch job every night on the primary database. Unfortunately, you accidentally ran the batch job twice, and you did not realize the mistake until the batch job completed for the second time. Ideally, you need to roll back the database to the point in time before the batch job began. A primary database that has a standby database with a time lag could help you to recover. You could fail over the standby database with the time lag and use it as the new primary database.

To create a standby database with a time lag, use the `DELAY` attribute of the `LOG_ARCHIVE_DEST_n` initialization parameter in the primary database initialization parameter file.

Note: If you define a delay for a destination that has real-time apply enabled, the delay is ignored

Although the redo data is still automatically transmitted from the primary database to the standby database and written to archived redo log files (and standby redo log files, if implemented), the log files are not immediately applied to the standby database. The log files are applied when the specified time interval expires.

This scenario uses a 4-hour time lag and covers the following topics:

- [Establishing a Time Lag on a Physical Standby Database](#)
- [Failing Over to a Physical Standby Database with a Time Lag](#)
- [Switching Over to a Physical Standby Database with a Time Lag](#)

Readers of this scenario are assumed to be familiar with the procedures for creating a typical standby database. The details were omitted from the steps outlined in this scenario. See [Chapter 3](#) for details about creating physical standby databases.

12.8.1 Establishing a Time Lag on a Physical Standby Database

To create a physical standby database with a time lag, modify the `LOG_ARCHIVE_DEST_n` initialization parameter on the primary database to set a delay for the standby database. The following is an example of how to add a 4-hour delay to the `LOG_ARCHIVE_DEST_n` initialization parameter:

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_2='SERVICE=stby DELAY=240';
```

The `DELAY` attribute indicates that the archived redo log files at the standby site are not available for recovery until the 4-hour time interval has expired. The time interval (expressed in minutes) starts when the archived redo log files are successfully archived at the standby site. The redo information is still sent to the standby database and written to the disk as normal.

See [Section 6.2.2](#) for a more information about establishing a time lag on physical and logical standby databases.

12.8.2 Failing Over to a Physical Standby Database with a Time Lag

A standby database configured to delay application of archived redo log files can be used to recover from user errors or data corruptions on the primary database. In most cases, you can query the time-delayed standby database to retrieve the data needed to repair the primary database (for example, to recover the contents of a mistakenly dropped table). In cases where the damage to the primary database is unknown or when the time required to repair the primary database is prohibitive, you can also consider failing over to a time-delayed standby database.

Assume that a backup file was inadvertently applied twice to the primary database and that the time required to repair the primary database is prohibitive. You choose to fail over to a physical standby database for which the application of archived redo log files is delayed. By doing so, you transition the standby database to the primary role at a point before the problem occurred, but you will likely incur some data loss. The following steps illustrate the process:

1. Initiate the failover by issuing the appropriate SQL statements on the time-delayed physical standby database:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE CANCEL;  
SQL> ALTER DATABASE ACTIVATE PHYSICAL STANDBY DATABASE;  
SQL> SHUTDOWN IMMEDIATE;  
SQL> STARTUP
```

The `ACTIVATE` statement immediately transitions the standby database to the primary role and makes no attempt to apply any additional redo data that might exist at the standby location. When using this statement, you must carefully balance the cost of data loss at the standby location against the potentially extended period of downtime required to fully repair the primary database.

2. Re-create all other standby databases in the configuration from a copy of this new primary database.

12.8.3 Switching Over to a Physical Standby Database with a Time Lag

All of the redo data is transmitted to the standby site as it becomes available. Therefore, even when a time delay is specified for a standby database, you can make the standby database current by overriding the delay using the `SQL ALTER DATABASE RECOVER MANAGED STANDBY` statement.

Note: To recover from a logical error, you must perform a failover instead of a switchover.

The following steps demonstrate how to perform a switchover to a time-delayed physical standby database that bypasses a time lag. For the purposes of this example, assume that the primary database is located in New York, and the standby database is located in Boston.

Step 1 Apply all of the archived redo log files to the original (time-delayed) standby database bypassing the lag.

Switchover will not begin until the standby database applies all of the archived redo log files. By lifting the delay using the `NODELAY` keyword, you allow the standby database to proceed without waiting for the specified time interval to pass before applying the archived redo log files.

Issue the following SQL statement to lift the delay:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE NODELAY
      2> DISCONNECT FROM SESSION THROUGH LAST SWITCHOVER;
```

Step 2 Stop read or update activity on the primary and standby databases.

You must have exclusive database access before beginning a switchover. Ask users to log off the primary and standby databases, or query the `V$SESSION` view to identify users that are connected to the databases and close all open sessions except the `SQL*Plus` session from which you are going to execute the switchover statement. See *Oracle Database Administrator's Guide* for more information about managing users.

Step 3 Switch the primary database to the physical standby role.

On the primary database (in New York), execute the following statement:

```
SQL> ALTER DATABASE COMMIT TO SWITCHOVER TO PHYSICAL STANDBY
      2> WITH SESSION SHUTDOWN;
```

This statement does the following:

- Closes the primary database, terminating any active sessions
- Transmits any unarchived redo log files and applies them to the standby database (in Boston)
- Adds an end-of-redo marker to the header of the last log file being archived
- Creates a backup of the current control file
- Converts the current control file into a standby control file

Step 4 Shut down and start up the former primary instance, and mount the database.

Execute the following statement on the former primary database (in New York):

```
SQL> SHUTDOWN NORMAL;
SQL> STARTUP MOUNT;
```

Step 5 Switch the original standby database to the primary role.

Issue the following SQL statement:

```
SQL> ALTER DATABASE COMMIT TO SWITCHOVER TO PRIMARY DATABASE;
```

Step 6 Shut down and restart the new primary database instance.

Issue the following SQL statements:

```
SQL> SHUTDOWN;
```

12.9 Recovering From a Network Failure

The following steps describe how to recover after a network failure.

Step 1 Identify the network failure.

The V\$ARCHIVE_DEST view contains the network error and identifies which standby database cannot be reached. On the primary database, execute the following SQL statement for the destination that experienced the network failure. For example:

```
SQL> SELECT DEST_ID, STATUS, ERROR FROM V$ARCHIVE_DEST WHERE DEST_ID = 2;
```

DEST_ID	STATUS	ERROR
2	ERROR	ORA-12224: TNS:no listener

The query results show there are errors archiving to the standby database, and the cause of the error is TNS:no listener. You should check whether or not the listener on the standby site is started. If the listener is stopped, then start it.

Step 2 Prevent the primary database from stalling.

If you cannot solve the network problem quickly, and if the standby database is specified as a mandatory destination, try to prevent the database from stalling by doing one of the following:

- Defer archiving to the mandatory destination:

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2 = DEFER;
```

When the network problem is resolved, you can enable the archive destination again:

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2 = ENABLE;
```

- Change the archive destination from mandatory to optional:

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_2 = 'SERVICE=standby1
2> OPTIONAL REOPEN=60';
```

When the network problem is resolved, you can change the archive destination from optional back to mandatory:

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_2 = 'SERVICE=standby1
2> MANDATORY REOPEN=60';
```

Step 3 Archive the current online redo log file.

On the primary database, archive the current online redo log file:

```
SQL> ALTER SYSTEM ARCHIVE LOG CURRENT;
```

When the network is back up again, log apply services can detect and resolve the archive gaps automatically when the physical standby database resumes Redo Apply.

12.10 Recovering After the NOLOGGING Clause Is Specified

In some SQL statements, the user has the option of specifying the NOLOGGING clause, which indicates that the database operation is not logged in the online redo log file. Even though the user specifies the clause, a redo record is still written to the online redo log file. However, there is no data associated with this record. This can result in log application or data access errors at the standby site and manual recovery might be required to resume applying log files.

Note: To avoid these problems, Oracle recommends that you always specify the `FORCE LOGGING` clause in the `CREATE DATABASE` or `ALTER DATABASE` statements. See the *Oracle Database Administrator's Guide*.

12.10.1 Recovery Steps for Logical Standby Databases

For logical standby databases, when SQL Apply encounters a redo record for an operation performed with the `NOLOGGING` clause, it skips over the record and continues applying changes from later records. Later, if an attempt is made to access one of the records that was updated with `NOLOGGING` in effect, the following error is returned: `ORA-01403 no data found`

To recover after the `NOLOGGING` clause is specified, re-create one or more tables from the primary database, as described in [Section 9.4.6](#).

Note: In general, use of the `NOLOGGING` clause is not recommended. Optionally, if you know in advance that operations using the `NOLOGGING` clause will be performed on certain tables in the primary database, you might want to prevent the application of SQL statements associated with these tables to the logical standby database by using the `DBMS_LOGSTDBY.SKIP` procedure.

12.10.2 Recovery Steps for Physical Standby Databases

When the archived redo log file is copied to the standby site and applied to the physical standby database, a portion of the datafile is unusable and is marked as being unrecoverable. When you either fail over to the physical standby database, or open the standby database for read-only access, and attempt to read the range of blocks that are marked as `UNRECOVERABLE`, you will see error messages similar to the following:

```
ORA-01578: ORACLE data block corrupted (file # 1, block # 2521)
ORA-01110: data file 1: '/oracle/dbs/stdby/tbs_1.dbf'
ORA-26040: Data block was loaded using the NOLOGGING option
```

To recover after the `NOLOGGING` clause is specified, you need to copy the datafile that contains the missing redo data from the primary site to the physical standby site. Perform the following steps:

Step 1 Determine which datafiles should be copied.

Follow these steps:

1. Query the primary database:

```
SQL> SELECT NAME, UNRECOVERABLE_CHANGE# FROM V$DATAFILE;
NAME                                     UNRECOVERABLE
-----
/oracle/dbs/tbs_1.dbf                    5216
/oracle/dbs/tbs_2.dbf                     0
/oracle/dbs/tbs_3.dbf                     0
/oracle/dbs/tbs_4.dbf                     0
4 rows selected.
```

2. Query the standby database:

```
SQL> SELECT NAME, UNRECOVERABLE_CHANGE# FROM V$DATAFILE;
NAME                                     UNRECOVERABLE
```

```

-----
/oracle/dbs/stdbby/tbs_1.dbf          5186
/oracle/dbs/stdbby/tbs_2.dbf          0
/oracle/dbs/stdbby/tbs_3.dbf          0
/oracle/dbs/stdbby/tbs_4.dbf          0
4 rows selected.

```

3. Compare the query results of the primary and standby databases.

Compare the value of the UNRECOVERABLE_CHANGE# column in both query results. If the value of the UNRECOVERABLE_CHANGE# column in the primary database is greater than the same column in the standby database, then the datafile needs to be copied from the primary site to the standby site.

In this example, the value of the UNRECOVERABLE_CHANGE# in the primary database for the tbs_1.dbf datafile is greater, so you need to copy the tbs_1.dbf datafile to the standby site.

Step 2 On the primary site, back up the datafile you need to copy to the standby site.

Issue the following SQL statements:

```

SQL> ALTER TABLESPACE system BEGIN BACKUP;
SQL> EXIT;
% cp tbs_1.dbf /backup
SQL> ALTER TABLESPACE system END BACKUP;

```

Step 3 Copy the datafile to the standby database.

Copy the datafile that contains the missing redo data from the primary site to location on the physical standby site where files related to recovery are stored.

Step 4 On the standby database, restart Redo Apply.

Issue the following SQL statement:

```

SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE DISCONNECT FROM SESSION;

```

You might get the following error messages (possibly in the alert log) when you try to restart Redo Apply:

```

ORA-00308: cannot open archived log 'standby1'
ORA-27037: unable to obtain file status
SVR4 Error: 2: No such file or directory
Additional information: 3
ORA-01547: warning: RECOVER succeeded but OPEN RESETLOGS would get error below
ORA-01152: file 1 was not restored from a sufficiently old backup
ORA-01110: data file 1: '/oracle/dbs/stdbby/tbs_1.dbf'

```

If you get the ORA-00308 error and Redo Apply does not terminate automatically, you can cancel recovery by issuing the following statement from another terminal window:

```

SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE CANCEL;

```

These error messages are returned when one or more log files in the archive gap have not been successfully applied. If you receive these errors, manually resolve the gaps, and repeat Step 4. See [Section 5.8.4](#) for information about manually resolving an archive gap.

12.10.3 Determining If a Backup Is Required After Unrecoverable Operations

If you performed unrecoverable operations on your primary database, determine if a new backup operation is required by following these steps:

1. Query the V\$DATAFILE view on the primary database to determine the **system change number (SCN)** or the time at which the Oracle database generated the most recent invalidated redo data.
2. Issue the following SQL statement on the primary database to determine if you need to perform another backup:

```
SELECT UNRECOVERABLE_CHANGE#,
       TO_CHAR (UNRECOVERABLE_TIME, 'mm-dd-yyyy hh:mi:ss')
FROM   V$DATAFILE;
```

3. If the query in the previous step reports an unrecoverable time for a datafile that is more recent than the time when the datafile was last backed up, then make another backup of the datafile in question.

See *Oracle Database Reference* for more information about the V\$DATAFILE view.

12.11 Resolving Archive Gaps Manually

An **archive gap** is a range of archived redo log files created whenever you are unable to apply the next archived redo log file generated by the primary database to the standby database. This section contains the following topics:

- [What Causes Archive Gaps?](#)
- [Determining If an Archive Gap Exists](#)
- [Manually Transmitting Log Files in the Archive Gap to the Standby Site](#)
- [Manually Applying Log Files in the Archive Gap to the Standby Database](#)

Note: Typically, archive gaps are resolved automatically without the need for manual intervention. See [Section 5.8](#) for more information about how log apply services automatically recover from gaps in the archived redo log files.

12.11.1 What Causes Archive Gaps?

An archive gap can occur whenever the primary database archives the current online redo log file locally, but the redo data is not archived at the standby site. Because the standby database requires the sequential application of log files, media recovery stops at the first missing log file encountered.

Archive gaps can occur in the following situations:

- [Creation of the Standby Database](#)
- [Shutdown of the Standby Database When the Primary Database Is Open](#)
- [Network Failure Prevents Transmission of Redo](#)

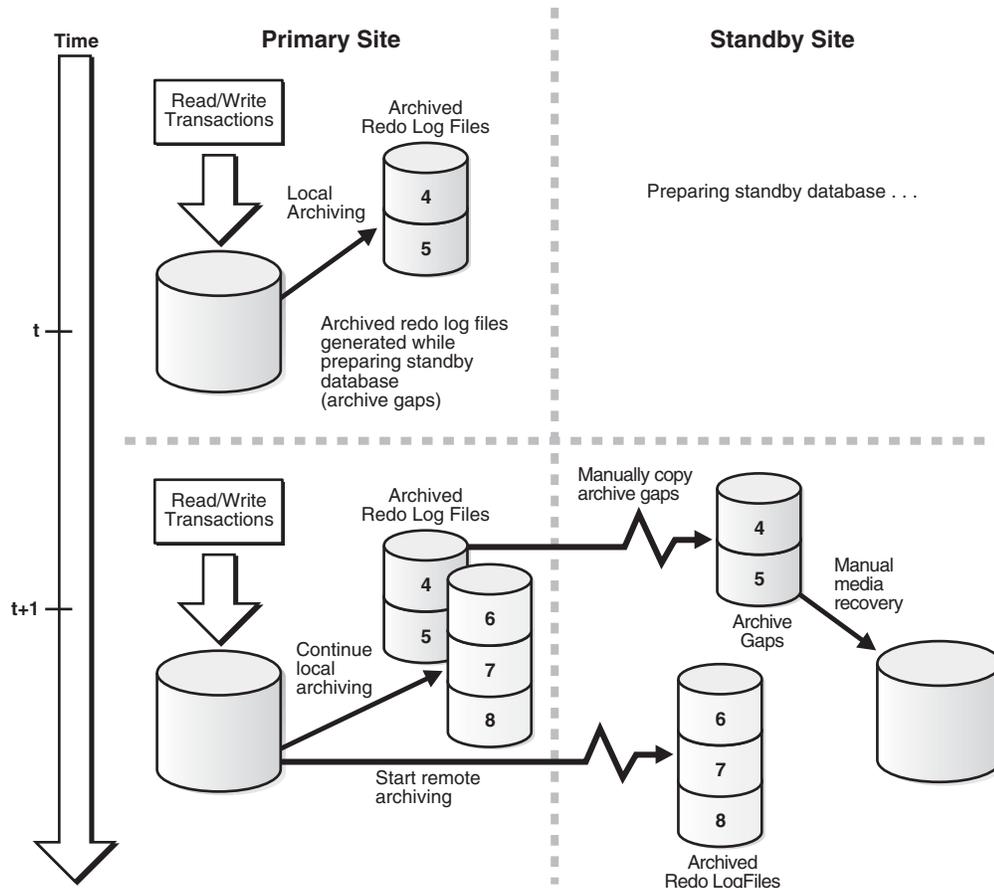
12.11.1.1 Creation of the Standby Database

One example of an archive gap occurs when you create the standby database from an old backup. For example, if the standby database is made from a backup that contains changes through log file 100, and the primary database currently contains changes

through log file 150, then the standby database requires that you apply log files 101 to 150. Another typical example of an archive gap occurs when you generate the standby database from a hot backup of an open database.

For example, assume the scenario illustrated in [Figure 12–8](#).

Figure 12–8 Manual Recovery of Archived Redo Log Files in an Archive Gap



The following steps occur:

1. You take a hot backup of primary database .
2. At time t , while you are busy configuring the network files, primary archives log files, sequences 4 and 5.
3. At time $t + 1$, you start the standby instance.
4. primary archives redo log files with sequences 6, 7, and 8 on the primary site, and transmits the redo to the standby site.

Archived redo log file sequences 4 and 5 are now part of an archive gap, and these log files must be applied to the standby database.

12.11.1.2 Shutdown of the Standby Database When the Primary Database Is Open

You might be required to shut down the standby database to resolve maintenance issues. For example, you must shut down the standby database when you change a control file parameter, such as `MAXDATAFILE`, in the primary database.

To avoid creating archive gaps, follow these rules:

- Start the standby databases and listeners *before* starting the primary database.
- Shut down the primary database *before* shutting down the standby database.

If you violate either of these two rules, then the standby database is down while the primary database is open and archiving. Consequently, the Oracle database can create an archive gap.

Note: If the standby site is specified as `MANDATORY` in one of the `LOG_ARCHIVE_DEST_n` parameters of the primary initialization parameter file, dynamically change it to `OPTIONAL` before shutting down the standby database. Otherwise, the primary database eventually stalls because it cannot archive its online redo log files.

12.11.1.3 Network Failure Prevents Transmission of Redo

If you maintain a Data Guard environment, and the network goes down, the primary database might continue to write to disk but be unable to send redo to the standby site. In this situation, archived redo log files accumulate as usual on the primary site, but the standby database is unaware of them.

See:

- [Section 5.7.2](#) for a detailed account of the significance of the `OPTIONAL` and `MANDATORY` attributes for standby archival
- [Section 12.9](#) for a related scenario

12.11.2 Determining If an Archive Gap Exists

To determine if there is an archive gap, query the `V$ARCHIVED_LOG` and `V$LOG` views. If an archive gap exists, the output of the query specifies the thread number and log sequence number of all log files in the archive gap. If there is *no* archive gap for a given thread, the query returns no rows.

Identify the log files in the archive gap

Query the `V$ARCHIVED_LOG` and `V$LOG` views on the standby database. For example, the following query shows there is a difference in the `RECD` and `SENT` sequence numbers for the destination specified by `DEST_ID=2`, indicating that there is a gap:

```
SQL> SELECT MAX(R.SEQUENCE#) LAST_SEQ_RECD, MAX(L.SEQUENCE#) LAST_SEQ_SENT FROM
2> V$ARCHIVED_LOG R, V$LOG L WHERE
3> R.DEST_ID=2 AND L.ARCHIVED='YES';
```

```
LAST_SEQ_RECD LAST_SEQ_SENT
-----
              7             10
```

Use the following query to determine the names of the archived redo log files on the local system that must be copied to the standby system that has the gap:

```
SQL> SELECT NAME FROM V$ARCHIVED_LOG WHERE THREAD#=1 AND DEST_ID=1 AND
2> SEQUENCE# BETWEEN 7 AND 10;
```

```
NAME
-----
/primary/thread1_dest/arcr_1_7.arc
/primary/thread1_dest/arcr_1_8.arc
/primary/thread1_dest/arcr_1_9.arc
```

```
/primary/thread1_dest/arcr_1_10.arc
```

12.11.3 Manually Transmitting Log Files in the Archive Gap to the Standby Site

After you have obtained the sequence numbers of the log files in the archive gap, you can obtain their filenames by querying the `V$ARCHIVED_LOG` view on the primary site. The archived redo log path names on the standby site are generated by the `STANDBY_ARCHIVE_DEST` and `LOG_ARCHIVE_FORMAT` parameters in the standby initialization parameter file.

If the standby database is on the same site as the primary database, or the standby database is on a remote site with a different directory structure than the primary database, the path names for the log files on the standby site cannot be the same as the path names of the log files archived by the primary database. Before transmitting the redo data to the standby site, determine the correct path names for the archived redo log files at the standby site.

To copy log files in an archive gap to the standby site

1. Review the list of archive gap log files that you obtained earlier. For example, assume you have the following archive gap:

THREAD#	LOW_SEQUENCE#	HIGH_SEQUENCE#
1	460	463
2	202	204
3	100	100

If a thread appears in the view, then it contains an archive gap. You need to copy log files from threads 1, 2, and 3.

2. Determine the path names of the log files in the archive gap that were transmitted by the primary database. After connecting to the primary database, issue a SQL query to obtain the name of a log file in each thread. For example, use the following SQL statement to obtain filenames of log files for thread 1:

```
SQL> SELECT NAME FROM V$ARCHIVED_LOG WHERE THREAD#=1 AND DEST_ID=1
2> AND SEQUENCE# > 459 AND SEQUENCE# < 464;
```

```
NAME
```

```
-----
/primary/thread1_dest/arcr_1_460.arc
/primary/thread1_dest/arcr_1_461.arc
/primary/thread1_dest/arcr_1_462.arc
/primary/thread1_dest/arcr_1_463.arc
4 rows selected
```

Perform similar queries for threads 2 and 3.

3. On the standby site, review the settings for `STANDBY_ARCHIVE_DEST` and `LOG_ARCHIVE_FORMAT` in the standby initialization parameter file. For example, you discover the following:

```
STANDBY_ARCHIVE_DEST = /standby/arc_dest/
LOG_ARCHIVE_FORMAT = log_%t_%s_%r.arc
```

These parameter settings determine the filenames of the archived redo log files at the standby site.

4. On the primary site, copy the log files in the archive gap from the primary site to the standby site, renaming them according to values for `STANDBY_ARCHIVE_`

DEST and LOG_ARCHIVE_FORMAT. For example, enter the following copy commands to copy the archive gap log files required by thread 1:

```
% cp /primary/thread1_dest/arcr_1_460.arc /standby/arc_dest/log_1_460.arc
% cp /primary/thread1_dest/arcr_1_461.arc /standby/arc_dest/log_1_461.arc
% cp /primary/thread1_dest/arcr_1_462.arc /standby/arc_dest/log_1_462.arc
% cp /primary/thread1_dest/arcr_1_463.arc /standby/arc_dest/log_1_463.arc
```

Perform similar commands to copy archive gap log files for threads 2 and 3.

5. On the standby site, if the LOG_ARCHIVE_DEST and STANDBY_ARCHIVE_DEST parameter values are *not* the same, then copy the archive gap log files from the STANDBY_ARCHIVE_DEST directory to the LOG_ARCHIVE_DEST directory. If these parameter values *are* the same, then you do not need to perform this step.

For example, assume the following standby initialization parameter settings:

```
STANDBY_ARCHIVE_DEST = /standby/arc_dest/
LOG_ARCHIVE_DEST = /log_dest/
```

Because the parameter values are different, copy the archived redo log files to the LOG_ARCHIVE_DEST location:

```
% cp /standby/arc_dest/* /log_dest/
```

When you initiate manual recovery, the Oracle database looks at the LOG_ARCHIVE_DEST value to determine the location of the log files.

Now that all required log files are in the STANDBY_ARCHIVE_DEST directory, you can proceed to [Section 12.11.4](#) to apply the archive gap log files to the standby database. See also [Section 8.5.4.4](#) and the V\$ARCHIVED_LOG view in [Chapter 16](#).

12.11.4 Manually Applying Log Files in the Archive Gap to the Standby Database

After you have copied the log files in the archive gap to the standby site, you can apply them using the RECOVER AUTOMATIC statement.

To apply the archived redo log files in the archive gap

1. Start up and mount the standby database (if it is not already mounted). For example, enter:

```
SQL> STARTUP MOUNT PFILE=/oracle/admin/pfile/initSTBY.ora
```

2. Recover the database using the AUTOMATIC option:

```
SQL> ALTER DATABASE RECOVER AUTOMATIC STANDBY DATABASE;
```

The AUTOMATIC option automatically generates the name of the next archived redo log file needed to continue the recovery operation.

After recovering the available log files, the Oracle database prompts for the name of a log file that does not exist. For example, you might see:

```
ORA-00308: cannot open archived log '/oracle/standby/standby_logs/arcr_1_540.arc'
ORA-27037: unable to obtain file status
SVR4 Error: 2: No such file or directory
Additional information: 3
Specify log: {<RET>=suggested | filename | AUTO | CANCEL}
```

3. Cancel recovery after the Oracle database applies the available log files by typing CTRL/C:

```
SQL> <CTRL/C>
Media recovery cancelled.
```

The following error messages are acceptable after recovery cancellation and do not indicate a problem:

```
ORA-01547: warning: RECOVER succeeded but OPEN RESETLOGS would get error below
ORA-01194: file 1 needs more recovery to be consistent
ORA-01110: data file 1: 'some_filename'
ORA-01112: media recovery not started
```

4. After you finish manually applying the missing log file, you can restart log apply services on the standby database, as follows:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE DISCONNECT FROM SESSION;
```

12.12 Creating a Standby Database That Uses OMF or ASM

[Chapter 3](#) and [Chapter 4](#) described how to create physical and logical standby databases. This section augments the discussions in those chapters with additional steps that must be performed if the primary database uses Oracle Managed Files (OMF) or Automatic Storage Management (ASM).

Note: The discussion in this section is presented at a level of detail that assumes the reader already knows how to create a physical standby database and is an experienced user of the RMAN, OMF, and ASM features. For more information, see:

- [Chapter 3](#), [Chapter 4](#), and [Appendix F](#) for information about creating physical and logical standby databases
 - *Oracle Database Administrator's Guide* for information about OMF and ASM
 - *Oracle Database Backup and Recovery Advanced User's Guide* and *Oracle Database Backup and Recovery Reference* for information about RMAN
-

Perform the following tasks to prepare for standby database creation:

1. Enable forced logging on the primary database.
2. Enable archiving on the primary database.
3. Set all necessary initialization parameters on the primary database.
4. Create an initialization parameter file for the standby database.
5. If the primary database is configured to use OMF, then Oracle recommends that the standby database be configured to use OMF, too. To do this, set the `DB_CREATE_FILE_DEST` and `DB_CREATE_ONLINE_LOG_DEST_n` initialization parameters to appropriate values. Maintenance and future role transitions are simplified if the same disk group names are used for both the primary and standby databases.
6. Set the `STANDBY_FILE_MANAGEMENT` initialization parameter to `AUTO`.
7. Configure Oracle Net, as required, to allow connections to the standby database.
8. Create a remote login password file for the standby database. Use the same password for the SYS account as on the primary database.

9. Start the standby database instance without mounting the control file.

Perform the following tasks to create the standby database:

1. If the standby database is going to use ASM, create an ASM instance if one does not already exist on the standby database system.
2. Use the RMAN BACKUP command to create a backup set that contains a copy of the primary database's datafiles, archived log files, and a standby control file.
3. Use the RMAN DUPLICATE ... FOR STANDBY command to copy the datafiles, archived redo log files and standby control file in the backup set to the standby database's storage area.

The DUPLICATE ... FOR STANDBY command performs the actual data movement at the standby instance. If the backup set is on tape, the media manager must be configured so that the standby instance can read the backup set. If the backup set is on disk, the backup pieces must be readable by the standby instance, either by making their primary path names available through NFS, or by copying them to the standby system and using RMAN CATALOG BACKUPPIECE command to catalog the backup pieces before restoring them.

After you successfully complete these steps, continue with the steps in [Section 3.2.7](#), to verify the configuration of the physical standby database.

To create a logical standby database, continue with the standby database creation process described in [Chapter 4](#), but with the following modifications:

1. For a logical standby database, setting the DB_CREATE_FILE_DEST parameter does not force the creation of OMF filenames. However, if this parameter was set on the primary database, it must also be set on the standby database.
2. After creating a logical standby control file on the primary system, do not use an operating system command to copy this file to the standby system. Instead, use the RMAN RESTORE CONTROLFILE command to restore a copy of the logical standby control file to the standby system.
3. If the primary database uses OMF files, use RMAN to update the standby database control file to use the new OMF files created on the standby database. To perform this operation, connect only to the standby database, as shown in the following example:

```
> RMAN TARGET sys/oracle@lstdby
RMAN> CATALOG START WITH '+stby_diskgroup';
RMAN> SWITCH DATABASE TO COPY;
```

After you successfully complete these steps, continue with the steps in [Section 4.2.5](#) to start, recover, and verify the logical standby database.

Part II

Reference

This part provides reference material to be used in conjunction with the Oracle Data Guard standby database features. For more complete reference material, refer to the Oracle Database 10g documentation set.

This part contains the following chapters:

- [Chapter 13, "Initialization Parameters"](#)
- [Chapter 14, "LOG_ARCHIVE_DEST_n Parameter Attributes"](#)
- [Chapter 15, "SQL Statements Relevant to Data Guard"](#)
- [Chapter 16, "Views Relevant to Oracle Data Guard"](#)

Initialization Parameters

This chapter describes the initialization parameters that affect databases in a Data Guard environment.

[Table 13–1](#) lists the initialization parameters and indicates if the parameter applies to the primary database role, the standby database role, or both. The table also includes notes and recommendations specific to setting the parameters in a Data Guard environment. *Oracle Database Reference* provides complete initialization parameter information, including how to update initialization parameters by issuing the `ALTER SYSTEM SET` or `ALTER SESSION` statements (for example, `ALTER SYSTEM SET LOG_ARCHIVE_TRACE`) or by editing the initialization parameter files. See the Oracle operating system-specific documentation for more information about setting initialization parameters.

Table 13–1 Initialization Parameters for Instances in a Data Guard Configuration

Parameter	Primary Role?	Standby Role?	Notes and Recommendations
<code>ARCHIVE_LAG_TARGET = seconds</code>	Yes	Physical only	Optional. Forces a log switch after the specified number of seconds elapses.
<code>COMPATIBLE = release_number.</code>	Yes	Logical and physical	Data Guard requires a minimum value of 9.2.0.1.0. Set to a minimum of 10.2.0.0 to use Oracle Database 10g new features. Specify the same value on the primary and standby databases if you expect to do a switchover. If the values differ, redo transport services may be unable to transmit redo data from the primary database to the standby databases. See Section 3.2.3 for an example. For rolling upgrades using SQL Apply, set this parameter according to the guidelines described in Section 11.4, "Prepare to Upgrade" .
<code>CONTROL_FILE_RECORD_KEEP_TIME = number_of_days</code>	Yes	Logical and physical	Optional. Use this parameter to avoid overwriting a reusable record in the control file (that contains needed information such as an archived redo log file) for the specified number of days (from 0 to 365). See Section 5.7.4 .
<code>CONTROL_FILES = 'control_file_name', control_file_name', ...')</code>	Yes	Logical and physical	Required. Specify the path name and filename for one or more control files. The control files must already exist on the database. Oracle recommends using 2 control files. If another copy of the current control file is available, then an instance can be easily restarted after copying the good control file to the location of the bad control file. See Section 3.2.3 for an example.

Table 13–1 (Cont.) Initialization Parameters for Instances in a Data Guard Configuration

Parameter	Primary Role?	Standby Role?	Notes and Recommendations
DB_FILE_NAME_CONVERT = (location_of_primary_database_datafile', 'location_of_standby_database_datafile_name', '...')	No	Physical only	Required if the standby database is on the same system as the primary database or if the directory where the datafiles are located on the standby system is different from the primary system. This parameter must specify paired strings. The first string is a sequence of characters to be looked for in a primary database filename. If that sequence of characters is matched, it is replaced by the second string to construct the standby database filename. You can specify multiple pairs of filenames. See also Example 3–3 .
DB_UNIQUE_NAME = unique_service_provider_name_for_this_database	Yes	Logical and physical	Recommended, but required if you specify the LOG_ARCHIVE_CONFIG parameter. Specifies a unique name for this database. This name does not change even if the primary and standby databases reverse roles. The DB_UNIQUE_NAME parameter defaults to the value of the DB_NAME parameter. See also the LOG_ARCHIVE_CONFIG parameter and Section 5.4.2 .
FAL_CLIENT = Oracle_Net_service_name	Yes	Physical only	Required if the FAL_SERVER parameter is specified. Specifies the Oracle Net service name used by the FAL server (typically the primary database) to refer to the FAL client (standby database). See Section 5.8.3 .
FAL_SERVER = Oracle_Net_service_name	No	Physical only	Required if the FAL_CLIENT parameter is specified. Specifies one or more Oracle Net service names for the databases from which this standby database can fetch (request) missing archived redo log files. See Section 5.8.3 .
INSTANCE_NAME	Yes	Logical and physical	Optional. If this parameter is defined and the primary and standby databases reside on the same host, specify a different name for the standby database than you specify for the primary database. See Section 3.2.3 for an example.
LOG_ARCHIVE_CONFIG= 'DG_CONFIG= (db_unique_name, db_unique_name, ...)	Yes	Logical and physical	Recommended. Specify the DG_CONFIG attribute to identify the DB_UNIQUE_NAME for the primary database and each standby database in the Data Guard configuration. The default value of this parameter enables the primary database to send redo data to remote destinations and enables standby databases to receive redo data. The DG_CONFIG attribute must be set to enable the dynamic addition of a standby database to a Data Guard configuration that has a Real Application Clusters primary database running in either maximum protection or maximum availability mode. See Section 5.4.2 .
LOG_ARCHIVE_DEST_n = {LOCATION=path_name SERVICE=service_name, attribute, attribute, ... }	Yes	Logical and physical	Required. Define up to ten (where <i>n</i> = 1, 2, 3, ... 10) destinations, each of which must specify either the LOCATION or SERVICE attribute. Specify a corresponding LOG_ARCHIVE_DEST_STATE_n parameter for every LOG_ARCHIVE_DEST_n parameter. See Section 5.2.2 and Chapter 14 for more information.

Table 13–1 (Cont.) Initialization Parameters for Instances in a Data Guard Configuration

Parameter	Primary Role?	Standby Role?	Notes and Recommendations
LOG_ARCHIVE_DEST_STATE_1 = {ENABLE DEFER ALTERNATE RESET}	Yes	Logical and physical	Required. Specify a LOG_ARCHIVE_DEST_STATE_1 parameter to enable or disable redo transport services to transmit redo data to the specified (or to an alternate) destination. Define a LOG_ARCHIVE_DEST_STATE_1 parameter for every LOG_ARCHIVE_DEST_1 parameter. See also Section 5.2.2 and Chapter 14 .
LOG_ARCHIVE_FORMAT=log%d_%t_%s_%r.arc	Yes	Logical and physical	Required if you specify the STANDBY_ARCHIVE_DEST parameter. These parameters are concatenated together to generate fully qualified archived redo log filenames on the standby database. See also Section 5.7.1 .
LOG_ARCHIVE_LOCAL_FIRST = [TRUE FALSE]	Yes	No	Optional. Specify to control when archiver processes (ARCn) transmit; either <i>after</i> (TRUE) the online redo log file was successfully archived to at least one local destination, or <i>at the same time</i> (FALSE) the online redo log file is being archived to local destinations. See also Section 5.3.1 .
LOG_ARCHIVE_MAX_PROCESSES = integer	Yes	Logical and physical	Optional. Specify the number (from 1 to 30) of archiver (ARCn) processes you want Oracle software to invoke initially. The default value is 4. See Section 5.3.1.2 for more information about ARCn processing.
LOG_ARCHIVE_MIN_SUCCEED_DEST = integer	Yes	No	Optional. Define the minimum number (from 1 to 10) of destinations that must receive redo data successfully before the log writer process on the primary database can reuse the online redo log file.
LOG_ARCHIVE_TRACE = integer	Yes	Logical and physical	Optional. Set this parameter to trace the transmission of redo data to the standby site. The valid integer values (0, 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, or 4096) are described in Appendix G .
LOG_FILE_NAME_CONVERT = 'location_of_primary_database_redo_logs', 'location_of_standby_database_redo_logs'	No	Logical and physical	Required when the standby database is on the same system as the primary database or when the directory structure where the log files are located on the standby site is different from the primary site. This parameter converts the path names of the primary database online redo log file to path names on the standby database. See Section 3.2.3 for an example.
PARALLEL_MAX_SERVERS = integer	Yes	Logical only	Required. Specify the maximum number of parallel servers working on the logical standby database. This parameter must not be set to a value less than 5 on a logical standby database. For best results, set PARALLEL_MAX_SERVERS to a minimum of 9.
REMOTE_LOGIN_PASSWORDFILE = {EXCLUSIVE SHARED}	Yes	Logical and physical	Required. Specify on the primary and all standby databases.
SHARED_POOL_SIZE = bytes	Yes	Logical and physical	Optional. Use to specify the system global area (SGA) to stage the information read from the online redo log files. The more SGA that is available, the more information that can be staged.
SORT_AREA_SIZE = bytes	Yes	Logical and physical	Optional. Increase the SORT_AREA_SIZE size (default size is 65536 bytes) to improve the efficiency of large sorts. See also Section 8.2 .

Table 13–1 (Cont.) Initialization Parameters for Instances in a Data Guard Configuration

Parameter	Primary Role?	Standby Role?	Notes and Recommendations
STANDBY_ARCHIVE_DEST= <i>filespec</i>	No	Logical and physical	Optional. Specify the location of archived redo log files received on the standby database from the primary database. The STANDBY_ARCHIVE_DEST initialization parameter overrides the directory location specified with the LOG_ARCHIVE_DEST_ <i>n</i> parameter. STANDBY_ARCHIVE_DEST and LOG_ARCHIVE_FORMAT are concatenated to generate fully qualified log filenames. See Section 5.7.1 .
STANDBY_FILE_MANAGEMENT = {AUTO MANUAL}	Yes	Physical only	Set the STANDBY_FILE_MANAGEMENT parameter to AUTO so that when datafiles are added to or dropped from the primary database, corresponding changes are made in the standby database without manual intervention. If the directory structures on the primary and standby databases are different, you must also set the DB_FILE_NAME_CONVERT initialization parameter to convert the filenames of one or more sets of datafiles on the primary database to filenames on the (physical) standby database. See Example 3–3 for more information and examples.
USER_DUMP_DEST = <i>directory_</i> <i>path_name_of_trace_file</i>	Yes	Logical and physical	Required if you specify the LOG_ARCHIVE_TRACE parameter. The USER_DUMP_DEST specifies the path name for a directory where the server will write debugging trace files. See Appendix G .

LOG_ARCHIVE_DEST_n Parameter Attributes

This chapter provides reference information for the attributes of the LOG_ARCHIVE_DEST_n initialization parameter. The following list shows the attributes:

AFFIRM and NOAFFIRM
ALTERNATE
ARCH and LGWR
DB_UNIQUE_NAME
DELAY
DEPENDENCY
LOCATION and SERVICE
MANDATORY and OPTIONAL
MAX_CONNECTIONS
MAX_FAILURE
NET_TIMEOUT
NOREGISTER
REOPEN
SYNC and ASYNC
TEMPLATE
VALID_FOR
VERIFY

Each LOG_ARCHIVE_DEST_n destination must contain either a LOCATION or SERVICE attribute to specify a local disk directory or a remotely accessed database, respectively. All other attributes are optional.

Note: Several attributes of the LOG_ARCHIVE_DEST_n initialization parameter have been deprecated. These attributes are supported for backward compatibility only and are documented in the *Oracle Database Reference*.

See Also: [Chapter 5](#) for more information about defining LOG_ARCHIVE_DEST_n destinations and setting up redo transport services.

AFFIRM and NOAFFIRM

Controls whether redo transport services use synchronous or asynchronous I/O to write redo data to disk:

- **AFFIRM**—specifies that all disk I/O to archived redo log files and standby redo log files is performed synchronously and completes successfully before the log writer process continues.
- **NOAFFIRM**—specifies that all disk I/O to archived redo log files and standby redo log files is performed asynchronously; the log writer process on the primary database does not wait until the disk I/O completes before continuing.

Category	AFFIRM	NOAFFIRM
Data type	Keyword	Keyword
Valid values	Not applicable	Not applicable
Default Value	Not applicable	Not applicable
Requires attributes ...	Not applicable	Not applicable
Conflicts with attributes ...	NOAFFIRM	AFFIRM
Corresponds to ...	AFFIRM and ASYNC_BLOCKS columns of the V\$ARCHIVE_DEST view	AFFIRM and ASYNC_BLOCKS columns of the V\$ARCHIVE_DEST view

Usage Notes

- These attributes are optional. If neither the **AFFIRM** nor the **NOAFFIRM** attribute is specified, the default is **NOAFFIRM**.
- The **AFFIRM** attribute specifies that all disk I/O to archived redo log files and standby redo log files is performed synchronously and must complete before the log writer process continues. The **AFFIRM** attribute:
 - Is one of the required attributes to ensure no data loss will occur if the primary database fails.
 - Can be specified with either the **LOCATION** or **SERVICE** attributes for archival operations to local or remote destinations.
 - Can potentially affect primary database performance, as follows:
 - * When you specify the **LGWR** and **AFFIRM** attributes, the log writer process synchronously writes the redo data to disk, control is not returned to the user until the disk I/O completes, and online redo log files on the primary database might not be reusable until archiving is complete.
 - * When you specify the **ARCH** and **AFFIRM** attributes, **ARC_n** processes synchronously write the redo data to disk, the archival operation might take longer, and online redo log files on the primary database might not be reusable until archiving is complete.
 - * When you specify the **ASYNC** and **AFFIRM** attributes, performance is not affected.

Note: When the primary database is in the maximum protection or maximum availability mode, destinations defined with the `LGWR` and `SYNC` attributes are automatically placed in `AFFIRM` mode.

- The `NOAFFIRM` attribute specifies that all disk I/O to archived redo log files and standby redo log files is performed asynchronously; the log writer process on the primary database does not wait until the disk I/O completes before continuing.
- The `AFFIRM` and `NOAFFIRM` attributes apply only to archived redo log files and standby redo log files on remote standby destinations and have no effect on disk I/O for the primary database's online redo log files.
- These attributes can be specified with either the `LOCATION` attribute for local destinations or with the `SERVICE` attribute for remote destinations.

See also: [SYNC](#) and [ASync](#) attributes on page 14-23

Examples

The following example shows the `AFFIRM` attribute for a remote destination.

```
LOG_ARCHIVE_DEST_3='SERVICE=stby1 LGWR SYNC AFFIRM'  
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

ALTERNATE

Specifies an alternate archiving destination to be used when the original destination fails.

Category	ALTERNATE=LOG_ARCHIVE_DEST_n
Data Type	String
Valid Value	A LOG_ARCHIVE_DEST_n destination
Default Value	None. If an alternate destination is not specified, then redo transport services do not automatically change to another destination.
Requires attributes ...	Not applicable
Conflicts with attributes ...	None ¹
Corresponds to ...	ALTERNATE and STATUS columns of the V\$ARCHIVE_DEST view

¹ If the REOPEN attribute is specified with a nonzero value, the ALTERNATE attribute is ignored. If the MAX_FAILURE attribute is also specified with a nonzero value, and the failure count exceeds the specified failure threshold, the ALTERNATE destination is enabled. Therefore, the ALTERNATE attribute does not conflict with a nonzero REOPEN attribute value.

Usage Notes

- The ALTERNATE attribute is optional. If an alternate destination is not specified, then redo transport services do not automatically change to another destination if the original destination fails.
- You can specify only one alternate destination for each LOG_ARCHIVE_DEST_n parameter, but several enabled destinations can share the same alternate destination.
- Ideally, an alternate destination should specify either:
 - A different disk location on the same local standby database system (shown in [Example 14-1](#) on page 14-5)
 - A different network route to the same standby database system (shown in [Example 14-2](#) on page 14-5)
 - A remote standby database system that closely mirrors that of the enabled destination
- If no enabled destinations reference an alternate destination, the alternate destination is implied to be deferred, because there is no automatic method of enabling the alternate destination. However, you can enable (or defer) alternate destinations at runtime using either ALTER SYSTEM.
- Any destination can be designated as an alternate destination, given the following restrictions:
 - At least one local mandatory destination is enabled.
 - The number of enabled destinations must meet the defined LOG_ARCHIVE_MIN_SUCCEED_DEST parameter value.
 - A destination cannot be its own alternate.

- Increasing the number of enabled destinations decreases the number of available alternate archiving destinations.
- When a destination fails, its alternate destination is enabled on the next archival operation. There is no support for enabling the alternate destination in the middle of the archival operation because that would require rereading already processed blocks, and so forth. This is identical to the REOPEN attribute behavior.
- If the REOPEN attribute is specified with a nonzero value, the ALTERNATE attribute is ignored unless the MAX_FAILURE attribute has a nonzero value. If the MAX_FAILURE and REOPEN attributes have nonzero values and the failure count exceeds the specified failure threshold, the ALTERNATE destination is enabled. Therefore, the ALTERNATE attribute does not conflict with a nonzero REOPEN attribute value.

Examples

In the sample initialization parameter file in [Example 14–1](#), LOG_ARCHIVE_DEST_1 automatically fails over to LOG_ARCHIVE_DEST_2 on the next archival operation if an error occurs or the device becomes full.

Example 14–1 Automatically Failing Over to an Alternate Destination

```
LOG_ARCHIVE_DEST_1='LOCATION=/disk1 MANDATORY ALTERNATE=LOG_ARCHIVE_DEST_2'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_2='LOCATION=/disk2 MANDATORY'
LOG_ARCHIVE_DEST_STATE_2=ALTERNATE
```

Notice in the example that a destination can also be in the ALTERNATE *state*, as specified with the LOG_ARCHIVE_DEST_STATE_*n* initialization parameter. The ALTERNATE state defers redo transport services from transmitting redo data to this destination until such time as another destination failure automatically enables this destination. [Section 5.2.2](#) provides more information about the LOG_ARCHIVE_DEST_STATE_*n* parameter.

Example 14–2 Defining an Alternate Oracle Net Service Name to the Same Standby Database

This example shows how to define an alternate Oracle Net service name to the same standby database.

```
LOG_ARCHIVE_DEST_1='LOCATION=/disk1 MANDATORY'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_2='SERVICE=stby1_path1 OPTIONAL ALTERNATE=LOG_ARCHIVE_DEST_3'
LOG_ARCHIVE_DEST_STATE_2=ENABLE
LOG_ARCHIVE_DEST_3='SERVICE=stby1_path2 OPTIONAL'
LOG_ARCHIVE_DEST_STATE_3=ALTERNATE
```

ARCH and LGWR

Specifies whether redo transport services use archiver processes (ARCH) or the log writer process (LGWR) to collect transaction redo data and transmit it to standby destinations. If neither the ARCH or LGWR attributes are specified, the default is ARCH.

Category	ARCH	LGWR
Data Type	Keyword	Keyword
Valid values	Not applicable	Not applicable
Default value	Not applicable	Not applicable
Requires attributes ...	None	None
Conflicts with attributes ...	LGWR, ASYNC, NET_TIMEOUT	ARCH
Corresponds to ...	ARCHIVER, PROCESS, and SCHEDULE columns of the V\$ARCHIVE_DEST view	ARCHIVER, PROCESS, and SCHEDULE columns of the V\$ARCHIVE_DEST view

Usage Notes

- These attributes are optional. If neither the ARCH or LGWR attribute is specified, the default is ARCH.
- Redo transport services use ARCH processes when the ARCH attribute is specified, and the log writer process when the LGWR attribute is specified.
By default, archiving is performed by ARCH processes; you must explicitly specify the LGWR attribute for redo transport services to use the LGWR process. Although you cannot specify both LGWR and ARCH processes for the same destination, you can choose to use the log writer process for some destinations, while archiver processes transmit redo data for other destinations.
- If you change a destination's current archival process (for example, from the ARCH process to the LGWR process), archival processing does not change until the next log switch occurs.

Example

The following example shows the LGWR attribute with the LOG_ARCHIVE_DEST_ *n* parameter. See [Section 5.3](#) for more examples.

```
LOG_ARCHIVE_DEST_3='SERVICE=denver LGWR'
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

DB_UNIQUE_NAME

Specifies a unique name for the database at this destination.

Category	DB_UNIQUE_NAME= <i>name</i>
Data Type	String
Valid values	The name must match the value that was defined for this database with the DB_UNIQUE_NAME parameter.
Default value	None
Requires attributes ...	None
Conflicts with attributes ...	None
Corresponds to ...	DB_UNIQUE_NAME column of the V\$ARCHIVE_DEST view

Usage Notes

- This attribute *is optional* if:
 - The LOG_ARCHIVE_CONFIG=DG_CONFIG initialization parameter is not specified.
 - This is a local destination (specified with the LOCATION attribute).
- This attribute *is required* if the LOG_ARCHIVE_CONFIG=DG_CONFIG initialization parameter is specified and if this is a remote destination (specified with the SERVICE attribute).
- Use the DB_UNIQUE_NAME attribute to clearly identify the relationship between a primary and standby databases. This attribute is particularly helpful if there are multiple standby databases in the Data Guard configuration.
- The name specified by the DB_UNIQUE_NAME must match one of the DB_UNIQUE_NAME values in the DG_CONFIG list. Redo transport services validate that the DB_UNIQUE_NAME attribute of the database at the specified destination matches the DB_UNIQUE_NAME attribute or the connection to that destination is refused.
- The name specified by the DB_UNIQUE_NAME attribute must match the name specified by the DB_UNIQUE_NAME initialization parameter of the database identified by the destination.

Example

In the following example, the DB_UNIQUE_NAME parameter specifies *boston* (DB_UNIQUE_NAME=*boston*), which is also specified with the DB_UNIQUE_NAME attribute on the LOG_ARCHIVE_DEST_1 parameter. The DB_UNIQUE_NAME attribute on the LOG_ARCHIVE_DEST_2 parameter specifies the *chicago* destination. Both *boston* and *chicago* are listed in the LOG_ARCHIVE_CONFIG=DG_CONFIG parameter.

```
DB_UNIQUE_NAME=boston
LOG_ARCHIVE_CONFIG='DG_CONFIG=(chicago,boston,denver) '
LOG_ARCHIVE_DEST_1='LOCATION=/arch1/
  VALID_FOR=(ALL_LOGFILES,ALL_ROLES)
  DB_UNIQUE_NAME=boston'
LOG_ARCHIVE_DEST_2='SERVICE=Sales_DR
  VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)
  DB_UNIQUE_NAME=chicago'
```

DELAY

Specifies a time lag between when redo data is archived on a standby site and when the archived redo log file is applied to the standby database.

Category	DELAY[= <i>minutes</i>]
Data Type	Numeric
Valid values	>=0 minutes
Default Value	30 minutes
Requires attributes ...	SERVICE
Conflicts with attributes ...	LOCATION
Corresponds to ...	DELAY_MINS and DESTINATION columns of the V\$ARCHIVE_DEST view

Usage Notes

- The DELAY attribute is optional. By default there is no delay.
- The DELAY attribute indicates the archived redo log files at the standby destination are not available for recovery until the specified time interval has expired. The time interval is expressed in minutes, and it starts when the redo data is successfully transmitted to, and archived at, the standby site.
- The DELAY attribute may be used to protect a standby database from corrupted or erroneous primary data. However, there is a tradeoff because during failover it takes more time to apply all of the redo up to the point of corruption.
- The DELAY attribute does not affect the transmittal of redo data to a standby destination.
- If you have real-time apply enabled, any delay that you set will be ignored.
- Changes to the DELAY attribute take effect the next time redo data is archived (after a log switch). In-progress archiving is not affected.
- You can override the specified delay interval at the standby site, as follows:
 - For a physical standby database:


```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE NODELAY;
```
 - For a logical standby database:


```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY NODELAY;
```

See Also: *Oracle Database SQL Reference* for more information about these ALTER DATABASE statements

Examples

You can use the DELAY attribute to set up a configuration where multiple standby databases are maintained in varying degrees of synchronization with the primary database. However, this protection incurs some overhead during failover, because it takes Redo Apply more time to apply all the redo up to the corruption point.

For example, assume primary database A has standby databases B and C. Standby database B is set up as the disaster recovery database and therefore has no time lag.

Standby database C is set up with a 2-hour delay, which is enough time to allow user errors to be discovered before they are propagated to the standby database.

The following example shows how to specify the DELAY attribute for this configuration:

```
LOG_ARCHIVE_DEST_1='LOCATION=/oracle/dbs/'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_2='SERVICE=stbyB LGWR SYNC AFFIRM'
LOG_ARCHIVE_DEST_STATE_2=ENABLE
LOG_ARCHIVE_DEST_3='SERVICE=stbyC DELAY=120'
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

Note: Alternatively, you can use Flashback Database to revert the database to a point-in-time or SCN in a different database incarnation as long as there is sufficient flashback log data. Using Flashback Database is described in *Oracle Database Backup and Recovery Basics*.

DEPENDENCY

Specify this attribute for a standby destination that will receive redo data through shared access to an archival location at another destination. A destination defined with this attribute has a dependency on another destination, which is specified by the `DEPENDENCY` attribute.

Category	<code>DEPENDENCY=LOG_ARCHIVE_DEST_n</code>
Data Type	String value
Valid values	Not applicable
Default Value	None
Requires attributes ...	<code>SERVICE</code>
Conflicts with attributes ...	<code>LOCATION, NOREGISTER</code>
Corresponds to ...	<code>DEPENDENCY</code> column of the <code>V\$ARCHIVE_DEST</code> view

Usage Notes

- The `DEPENDENCY` attribute is optional. If you do not specify the `DEPENDENCY` attribute, redo transport services transmit redo data directly to the destination.
- The `DEPENDENCY` attribute can be specified on a physical standby database or a logical standby database.
 - **See Also:** [Section 5.7.5, "Sharing a Log File Destination Among Multiple Standby Databases"](#)
- The availability of redo data at a dependent destination relies on the success or failure of redo transmission to the destination that is specified by the `DEPENDENCY` attribute.
- The `DEPENDENCY` attribute has the following restrictions:
 - Only standby destinations can have a dependency.
 - The destination specified by the `DEPENDENCY` attribute can use either the `LOCATION` or `SERVICE` attribute.
 - The `DEPENDENCY` attribute cannot be modified at the session level.
- When one or more destinations are dependent on the same destination, all attributes specified for the dependent destinations still apply. It appears as if the archival operation was performed for each destination, when only one archival operation actually occurred.

For example, consider that two standby databases are dependent upon the same destination. You can specify different `DELAY` attributes for each destination, which enables you to maintain a staggered time lag between the primary database and each standby database.

Similarly, a dependent destination can specify an `ALTERNATE` destination, which itself might or might not be dependent on the same parent destination.
- A zero data-loss failover cannot be performed to a dependent destination.

Examples

One reason to use the `DEPENDENCY` attribute is if two standby databases reside on the same system. The parent and child standby databases can be any mix of physical and logical standby databases. For example:

```
# Set up the mandatory local destination:
#
LOG_ARCHIVE_DEST_1='LOCATION=/oracle/dbs/ MANDATORY'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
#
# Set up the remote standby database that will receive the redo data:
#
LOG_ARCHIVE_DEST_2='SERVICE=dest2 OPTIONAL'
LOG_ARCHIVE_DEST_STATE_2=ENABLE
#
# Set up the remote standby database that resides on the same system as, and is
# dependent on, the first standby database:
#
LOG_ARCHIVE_DEST_3='SERVICE=dest3 DEPENDENCY=LOG_ARCHIVE_DEST_2 OPTIONAL'
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

LOCATION and SERVICE

Each destination *must* specify either the `LOCATION` or the `SERVICE` attribute to identify either a local disk directory or a remote database destination where redo transport services can transmit redo data.

Category	<code>LOCATION=local_disk_directory</code> or <code>USE_DB_RECOVERY_FILE_DEST</code>	<code>SERVICE=net_service_name</code>
Data type	String value	String value
Valid values	Not applicable	Not applicable
Default Value	None	None
Requires attributes ...	Not applicable	Not applicable
Conflicts with attributes ...	<code>SERVICE</code> , <code>DELAY</code> , <code>DEPENDENCY</code> , <code>NOREGISTER</code> , <code>ASYNC</code> , <code>TEMPLATE</code> , <code>NET_TIMEOUT</code>	<code>LOCATION</code>
Corresponds to ...	<code>DESTINATION</code> and <code>TARGET</code> columns of the <code>V\$ARCHIVE_DEST</code> view	<code>DESTINATION</code> and <code>TARGET</code> columns of the <code>V\$ARCHIVE_DEST</code> view

Usage Notes

- Either the `LOCATION` or the `SERVICE` attribute must be specified. There is no default.
- If you are specifying multiple attributes, specify the `LOCATION` or `SERVICE` attribute first in the list of attributes.
- You must specify at least one local disk directory with the `LOCATION` attribute. This ensures the local archived redo log files are accessible should media recovery of the primary database be necessary. You can specify up to nine additional local or remote destinations. Specifying remote destinations with the `SERVICE` attribute ensures Data Guard can maintain a transactionally consistent remote copy of the primary database for disaster recovery.
- For the `LOCATION` attribute, you can specify one of the following:
 - `LOCATION=local_disk_directory`
This specifies a unique directory path name for a disk directory on the system that hosts the database. This is the local destination for archived redo log files.
 - `LOCATION=USE_DB_RECOVERY_FILE_DEST`
To configure a flash recovery area, you specify the directory or Oracle Storage Manager disk group that will serve as the flash recovery area using the `DB_RECOVERY_FILE_DEST` initialization parameter. If no local destinations are defined and a flash recovery area has been configured, Data Guard implicitly uses the `LOG_ARCHIVE_DEST_10` destination for the flash recovery area. See [Section 5.2.3](#) for more information about flash recovery areas.
- When you specify a `SERVICE` attribute:
 - You identify remote destinations by specifying the `SERVICE` attribute with a valid Oracle Net service name (`SERVICE=net_service_name`) that identifies the remote Oracle database instance to which the redo data will be sent.

The Oracle Net service name that you specify with the `SERVICE` attribute is translated into a connection descriptor that contains the information necessary for connecting to the remote database.

See Also: *Oracle Database Net Services Administrator's Guide* for details about setting up Oracle Net service names

- Transmitting redo data to a remote destination requires a network connection and an Oracle database instance associated with the remote destination to receive the incoming redo data.
- To verify the current settings for `LOCATION` and `SERVICE` attributes, query the `V$ARCHIVE_DEST` fixed view:
 - The `TARGET` column identifies if the destination is local or remote to the primary database.
 - The `DESTINATION` column identifies the values that were specified for a destination. For example, the destination parameter value specifies the Oracle Net service name identifying the remote Oracle instance where the archived redo log files are located.

Examples

Example 1 Specifying the `LOCATION` Attribute

```
LOG_ARCHIVE_DEST_2='LOCATION=/disk1/oracle/oradata/payroll/arch/'  
LOG_ARCHIVE_DEST_STATE_2=ENABLE
```

Example 2 Specifying the `SERVICE` Attribute

```
LOG_ARCHIVE_DEST_3='SERVICE=stby1'  
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

MANDATORY and OPTIONAL

Specifies the policy for reusing online redo log files:

- **MANDATORY**—specifies that filled online log files must be successfully archived to the destination before they can be reused.
- **OPTIONAL**—specifies that successful archival to the destination is not required before the online redo log file can be made available for reuse.

Category	MANDATORY	OPTIONAL
Data type	Keyword	Keyword
Valid values	Not applicable	Not applicable
Default value	Not applicable	Not applicable
Requires attributes ...	Not applicable	Not applicable
Conflicts with attributes ...	Optional	Mandatory
Corresponds to ...	BINDING column of the V\$ARCHIVE_DEST view	BINDING column of the V\$ARCHIVE_DEST view

Usage Notes

- If neither the **MANDATORY** nor the **OPTIONAL** attribute is specified, the default is **OPTIONAL**.
 At least one destination must succeed even if all destinations are designated as optional. If a destination is *optional*, archiving to that destination may fail, yet the online redo log file is available for reuse and may be overwritten eventually. If the archival operation of a *mandatory* destination fails, online redo log files cannot be overwritten.
- The `LOG_ARCHIVE_MIN_SUCCEED_DEST=n` parameter (where *n* is an integer from 1 to 10) specifies the number of destinations that must archive successfully before the log writer process can overwrite the online redo log files.
 All **MANDATORY** destinations and non-standby **OPTIONAL** destinations contribute to satisfying the `LOG_ARCHIVE_MIN_SUCCEED_DEST=n` count. If the value set for the `LOG_ARCHIVE_MIN_SUCCEED_DEST` parameter (that defines the minimum number of destinations that must receive redo data successfully before the log writer process on the primary database can reuse the online redo log file) is met, the online redo log file is available for reuse. For example, you can set the parameter as follows:

```
# Database must archive to at least two locations before
# overwriting the online redo log files.
LOG_ARCHIVE_MIN_SUCCEED_DEST = 2
```
- You must have at least one local destination, which you can declare **OPTIONAL** or **MANDATORY**.
 At least one local destination is operationally treated as mandatory, because the minimum value for the `LOG_ARCHIVE_MIN_SUCCEED_DEST` parameter is 1.
- The failure of any mandatory destination, including a mandatory standby destination, makes the `LOG_ARCHIVE_MIN_SUCCEED_DEST` parameter irrelevant.

- The LOG_ARCHIVE_MIN_SUCCEED_DEST parameter value cannot be greater than the number of mandatory destinations plus the number of optional local destinations.
- These attributes do not affect the data protection mode for the destination.
- The BINDING column of the V\$ARCHIVE_DEST fixed view specifies how failure affects the archival operation

Examples

The following example shows the MANDATORY attribute:

```
LOG_ARCHIVE_DEST_1='LOCATION=/arch/dest MANDATORY'  
LOG_ARCHIVE_DEST_STATE_1=ENABLE  
LOG_ARCHIVE_DEST_3='SERVICE=denvver MANDATORY'  
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

MAX_CONNECTIONS

Specifies the maximum number of network connections that will be used to perform remote archival to the destination. If the `MAX_CONNECTIONS` attribute is set to a value greater than 1, redo transport services use multiple network connections to perform remote archivals. Each of these connections use a separate archiver (`ARC n`) process.

Category	Description
Data type	Integer
Valid values	1 to 5
Default value	1
Requires attributes ...	None
Conflicts with attributes ...	None
Corresponds to ...	<ul style="list-style-type: none"> ■ <code>MAX_CONNECTIONS</code> column of the <code>V\$ARCHIVE_DEST</code> view of the primary database ■ <code>LOG_ARCHIVE_MAX_PROCESSES</code> and <code>PARALLEL_MAX_SERVERS</code> initialization parameters

Usage Notes

- The `MAX_CONNECTIONS` attribute is optional. By default redo transport services use a single network connection to perform a remote archival.
If the `MAX_CONNECTIONS` attribute is set to a value greater than 1, redo transport services use multiple network connections to perform a remote archival to the destination.
- The `LOG_ARCHIVE_MAX_PROCESSES` and `PARALLEL_MAX_SERVERS` initialization parameters are related to the `MAX_CONNECTIONS` attribute and affect the actual number of `ARC n` processes used by an instance.
For example, if the total of `MAX_CONNECTIONS` attributes on all destinations exceeds the value of `LOG_ARCHIVE_MAX_PROCESSES`, then Data Guard will use as many `ARC n` processes as possible but the number of connections may be less than is specified by `MAX_CONNECTIONS`.

Examples

The following example shows the `MAX_CONNECTIONS` attribute:

```
LOG_ARCHIVE_DEST_1='LOCATION=/arch/dest'  
LOG_ARCHIVE_DEST_STATE_1=ENABLE  
LOG_ARCHIVE_DEST_3='SERVICE=denvver MAX_CONNECTIONS=3'  
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

MAX_FAILURE

Controls the consecutive number of times redo transport services attempt to reestablish communication and transmit redo data to a failed destination before the primary database gives up on the destination.

Category	MAX_FAILURE= <i>count</i>
Data type	Numeric
Valid value	>=0
Default value	None
Requires attributes ...	REOPEN
Conflicts with attributes ...	None
Corresponds to	MAX_FAILURE, FAILURE_COUNT, and REOPEN_SECS columns of the V\$ARCHIVE_DEST view

Usage Notes

- The MAX_FAILURE attribute is optional. By default, there are an unlimited number of archival attempts to the failed destination.
- This attribute is useful for providing failure resolution for destinations to which you want to retry transmitting redo data after a failure, but not retry indefinitely.
- When you specify the MAX_FAILURE attribute, you must also set the REOPEN attribute. Once the specified number of consecutive attempts is exceeded, the destination is treated as if the REOPEN attribute was not specified.
- You can view the failure count in the FAILURE_COUNT column of the V\$ARCHIVE_DEST fixed view. The related column REOPEN_SECS identifies the REOPEN attribute value.

Note: Once the failure count for the destination reaches the specified MAX_FAILURE attribute value, the only way to reuse the destination is to modify the MAX_FAILURE attribute value or any attribute. This has the effect of resetting the failure count to zero (0).

- The failure count is reset to zero (0) whenever the destination is modified by an ALTER SYSTEM SET statement. This avoids the problem of setting the MAX_FAILURE attribute to a value less than the current failure count value.
- Once the failure count is greater than or equal to the value set for the MAX_FAILURE attribute, the REOPEN attribute value is implicitly set to zero, which causes redo transport services to transport redo data to an alternate destination (defined with the ALTERNATE attribute) on the next archival operation.
- Redo transport services attempt to archive to the failed destination indefinitely if you do not specify the MAX_FAILURE attribute (or if you specify MAX_FAILURE=0), and you specify a nonzero value for the REOPEN attribute. If the destination has the MANDATORY attribute, the online redo log file is not reusable until it has been archived to this destination.

Examples

The following example allows redo transport services up to three consecutive archival attempts, tried every 5 seconds, to the `arc_dest` destination. If the archival operation fails after the third attempt, the destination is treated as if the `REOPEN` attribute was not specified.

```
LOG_ARCHIVE_DEST_1='LOCATION=/arc_dest REOPEN=5 MAX_FAILURE=3'  
LOG_ARCHIVE_DEST_STATE_1=ENABLE
```

NET_TIMEOUT

Specifies the number of seconds the log writer process on the primary system waits for status from the network server (LNS n) process before terminating the network connection.

Category	NET_TIMEOUT= <i>seconds</i>
Data type	Numeric
Valid values	1 ¹ to 1200
Default value	180 seconds
Requires attributes ...	LGWR with SYNC
Conflicts with attributes ...	ARCH, LOCATION, LGWR with ASYNC ²
Corresponds to ...	NET_TIMEOUT column of the V\$ARCHIVE_DEST view of the primary database

¹ Although a minimum value of 1 second is allowed, Oracle recommends 8 to 10 seconds as a minimum to avoid *false errors* and disconnection from the standby database.

² If you specify the LGWR and ASYNC attributes, redo transport services ignore them; no error is returned.

Usage Notes

- The NET_TIMEOUT attribute is optional. However, if you do not specify the NET_TIMEOUT attribute it will be set to 180 seconds, but the primary database can potentially stall. To avoid this situation, specify a small, nonzero value for the NET_TIMEOUT attribute so the primary database can continue operation after the user-specified timeout interval expires when waiting for status from the network server.
- The NET_TIMEOUT attribute is used only when the log writer process transmits redo data using a network server (LNS n) process.
- The log writer process waits for the specified amount of time to receive status about the network I/O. If there is a possible network disconnection, even one that was terminated due to a network timeout, the log writer process automatically tries to reconnect to the standby database to resolve network brownouts and false network terminations. Typically, except when the network is physically broken, the log writer process can successfully reconnect to the network. The reconnection attempts continue for a period of time that depends on the following factors:
 - The value of the NET_TIMEOUT attribute on the primary database.
 - The protection mode of the primary database, which determines the maximum amount of time that the reconnection will take. Use the following time estimates as a guideline for how long the log writer process will try to reconnect to the standby database:
 - In maximum protection mode, the log writer process tries to reconnect for approximately 5 minutes.
 - In maximum availability mode, the log writer process tries to reconnect within the NET_TIMEOUT value.

For example, a primary database operating in the maximum availability protection mode with a NET_TIMEOUT attribute value set to 60 seconds could actually take a

minimum of 1 minute to connect or up to 3 minutes to terminate the connection to the standby database.

Caution: Be careful to specify a reasonable value for the NET_TIMEOUT attribute when running in maximum protection mode. A *false* network failure detection might cause the primary instance to shut down.

- Without careful coordination of the timeout parameter values on the primary and standby systems, it is possible that the primary system might detect a network problem and disconnect, while the standby database might not recognize the network disconnection if its default network timeout values are too high. If the network timers are not set up properly, subsequent attempts by the log writer process on the primary database to attach to the standby database will fail because the standby database has not yet timed out and the broken network connection still appears to be valid. See *Oracle Database Net Services Administrator's Guide*.

Examples

The following example shows how to specify a 40-second network timeout value on the primary database with the NET_TIMEOUT attribute.

```
LOG_ARCHIVE_DEST_2='SERVICE=stby1 LGWR NET_TIMEOUT=40 SYNC'  
LOG_ARCHIVE_DEST_STATE_2=ENABLE
```

NOREGISTER

Indicates that the location of the archived redo log file should not be recorded at the corresponding destination.

Category	NOREGISTER
Data type	Keyword
Valid values	Not applicable
Default value	Not applicable
Requires attributes ...	SERVICE
Conflicts with attributes ...	LOCATIONt
Corresponds to ...	DESTINATION and TARGET columns of the V\$ARCHIVE_DEST view

Usage Notes

- The NOREGISTER attribute *is optional* if the standby database destination is a part of a Data Guard configuration.
- The NOREGISTER attribute *is required* if the destination is not part of a Data Guard configuration.
- This attribute pertains to remote destinations only. The location of each archived redo log file is always recorded in the primary database control file.

Examples

The following example shows the NOREGISTER attribute:

```
LOG_ARCHIVE_DEST_5='NOREGISTER'
```

REOPEN

Specifies the minimum number of seconds before redo transport services should try to reopen a failed destination.

Category	REOPEN [=seconds]
Data Type	Numeric
Valid values	>=0 seconds
Default Value	300 seconds
Requires attributes ...	None
Conflicts with attributes ...	Not applicable
Corresponds to ...	REOPEN_SECS and MAX_FAILURE columns of the V\$ARCHIVE_DEST view

Usage Notes]

- The REOPEN attribute is optional.
- Redo transport services attempt to reopen failed destinations at log switch time.
- Redo transport services check if the time of the last error plus the REOPEN interval is less than the current time. If it is, redo transport services attempt to reopen the destination.
- REOPEN applies to all errors, not just connection failures. These errors include, but are not limited to, network failures, disk errors, and quota exceptions.
- If you specify REOPEN for an OPTIONAL destination, it is possible for the Oracle database to overwrite online redo log files if there is an error. If you specify REOPEN for a MANDATORY destination, redo transport services will stall the primary database when it is not possible to successfully transmit redo data. When this situation occurs, consider the following options:
 - Change the destination by deferring the destination, specifying the destination as optional, or changing the SERVICE attribute value.
 - Specify an alternate destination.
 - Disable the destination.

See Also: [Section 5.5, "What to Do If Errors Occur"](#) for more information about using the REOPEN, MAX_FAILURES, and ALTERNATE attributes to specify what actions are to be taken when archival processing to a destination fails

Examples

The following example shows the REOPEN attribute.

```
LOG_ARCHIVE_DEST_3='SERVICE=stby1 MANDATORY REOPEN=60'
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

SYNC and ASYNC

Specifies that network I/O is to be done synchronously (SYNC) or asynchronously (ASYNC) when archival is performed using the log writer process (LGWR).

Note: When the primary database is in maximum protection mode or maximum availability mode, destinations archiving to standby redo log files and using the log writer process are automatically placed in SYNC mode.

Category	SYNC	ASYNC
Data type	Keyword	Numeric
Valid values	Not applicable	Not applicable
Default value	Not applicable	None
Requires attributes ...	None	LGWR
Conflicts with attributes ...	ASYNC	SYNC, LOCATION, ARCH
Corresponds to ...	TRANSMIT_MODE column of the V\$ARCHIVE_DEST view	TRANSMIT_MODE and ASYNC_BLOCKS columns of the V\$ARCHIVE_DEST view

Usage Notes

- The SYNC and ASYNC attributes are optional:
 - When you specify the LGWR attribute but you do not specify either the SYNC or ASYNC attribute, the default is SYNC.
 - When you specify the ARCH attribute, only the SYNC attribute is valid. An error message is returned if you specify the ARCH and ASYNC attributes together.
- The SYNC attribute specifies that network I/O is to be performed synchronously for the destination, which means that once the I/O is initiated, the log writer process waits for the I/O to complete before continuing. The SYNC attribute is one requirement for setting up a no-data-loss environment, because it ensures the redo records are successfully transmitted to the standby destination before continuing.
- The ASYNC attribute specifies that network I/O is to be performed asynchronously for the destination.

Examples

The following example shows the SYNC attribute with the LOG_ARCHIVE_DEST_n parameter.

```
LOG_ARCHIVE_DEST_3='SERVICE=stby1 LGWR SYNC'
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

TEMPLATE

Defines a directory specification and format template for names of archived redo log files at the destination. The template is used to generate a filename that is different from the default filename format defined by the `STANDBY_ARCHIVE_DEST` and `LOG_ARCHIVE_FORMAT` initialization parameters at the standby destination.

Category	TEMPLATE= <i>filename_template_%t_%s_%r</i>
Data Type	String value
Valid values	Not applicable
Default Value	None
Requires attributes ...	SERVICE
Conflicts with attributes ...	LOCATION
Corresponds to ...	REMOTE_TEMPLATE and REGISTER columns of the V\$ARCHIVE_DEST view

Usage Notes

- The `TEMPLATE` attribute is optional. If `TEMPLATE` is not specified, archived redo logs are named using the value of the `STANDBY_ARCHIVE_DEST` and `LOG_ARCHIVE_FORMAT` initialization parameters.
- The `TEMPLATE` attribute overrides the `STANDBY_ARCHIVE_DEST` and `LOG_ARCHIVE_FORMAT` initialization parameter settings at the remote archival destination.
- The `TEMPLATE` attribute is valid only with remote destinations (that is, destinations specified with the `SERVICE` attribute).
- If specified for a destination that also specifies the `LGWR` attribute, rearchiving by the `ARCn` process does not use the `TEMPLATE` filename specification.
- The value you specify for *filename_template* must contain the `%s`, `%t`, and `%r` directives described in [Table 14-1](#).

Table 14-1 Directives for the TEMPLATE Attribute

Directive	Description
<code>%t</code>	Substitute the instance thread number.
<code>%T</code>	Substitute the instance thread number, zero filled.
<code>%s</code>	Substitute the log file sequence number.
<code>%S</code>	Substitute the log file sequence number, zero filled.
<code>%r</code>	Substitute the resetlogs ID.
<code>%R</code>	Substitute the resetlogs ID, zero filled.

- The *filename_template* value is transmitted to the standby destination, where it is translated and validated before creating the filename.

Examples

In the following example, `prmy1` transmits redo data to the remote destination, `stby1`. The `TEMPLATE` attribute indicates that `stby1` is located in the directory `/usr/oracle/prmy1` with the `p1_thread#_sequence#_resetlogs.dbf` filename format.

```
LOG_ARCHIVE_DEST_1='SERVICE=boston MANDATORY REOPEN=5
    TEMPLATE=/usr/oracle/prmy1/p1_%t_%s_%r.dbf'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
```

VALID_FOR

Specifies when redo transport services transmit redo data to a destination, based on the following factors:

- Whether the database is *currently* running in the primary or the standby role
- Whether online redo log files, standby redo log files, or both are *currently* being archived on the database at this destination

Category	VALID_FOR=(redo_log_type, database_role)
Data Type	String value
Valid values	Not applicable
Default Value	VALID_FOR=(ALL_LOGFILES, ALL_ROLES) ¹
Requires attributes ...	None
Conflicts with attributes ...	None
Corresponds to ...	VALID_NOW, VALID_TYPE, and VALID_ROLE columns in the V\$ARCHIVE_DEST view

¹ Do not use the default value, VALID_FOR=(ALL_LOGFILES, ALL_ROLES), for logical standby databases. See [Section 5.4.1](#) and the scenario in [Section 12.1.2](#) for more information.

Usage Notes

- The VALID_FOR attribute is optional. However, Oracle recommends that you define a VALID_FOR attribute for each destination so that your Data Guard configuration operates properly after a role transition.

Note: Although the (ALL_LOGFILES, ALL_ROLES) keyword pair is *the default*, it is not appropriate for every destination. For example, if the destination is a logical standby database, which is an open database that is creating its own redo data, the redo data being transmitted by redo transport services could potentially overwrite the logical standby database's local online redo log files.

- To configure these factors for each LOG_ARCHIVE_DEST_ *n* destination, you specify this attribute with a pair of keywords: VALID_FOR=(redo_log_type, database_role):
 - The redo_log_type keyword identifies the destination as valid for archiving one of the following:
 - ONLINE_LOGFILE—This destination is valid only when archiving online redo log files.
 - STANDBY_LOGFILE—This destination is valid only when archiving standby redo log files.
 - ALL_LOGFILES— This destination is valid when archiving either online redo log files or standby redo log files.
 - The database_role keyword identifies the role in which this destination is valid for archiving:

- PRIMARY_ROLE—This destination is valid only when the database is running in the primary role.
 - STANDBY_ROLE—This destination is valid only when the database is running in the standby role.
 - ALL_ROLES—This destination is valid when the database is running in either the primary or the standby role.
- The following table shows the VALID_FOR attribute values and the roles in which each might be used.

Table 14–2 VALID_FOR Attribute Values

VALID_FOR Definition	Primary Role	Physical Standby Role	Logical Standby Role
ONLINE_LOGFILE, PRIMARY_ROLE	Active	Inactive	Invalid
ONLINE_LOGFILE, STANDBY_ROLE	Inactive	Invalid	Active
ONLINE_LOGFILE, ALL_ROLES	Active	Invalid	Active
STANDBY_LOGFILE, PRIMARY_ROLE	Error	Error	Error
STANDBY_LOGFILE, STANDBY_ROLE	Invalid	Active	Active
STANDBY_LOGFILE ALL_ROLES	Invalid	Active	Active
ALL_LOGFILES, PRIMARY_ROLE	Active	Inactive	Invalid
ALL_LOGFILES, STANDBY_ROLE	Invalid	Active	Active
ALL_LOGFILES, ALL_ROLES	Active	Active	Active

Note: The VALID_FOR=(STANDBY_LOGFILE, PRIMARY_ROLE) keyword pair is not possible; although it is valid to configure standby redo log files on a primary database, a database that is running in the primary role cannot use standby redo log files.

- If you do not specify the VALID_FOR attribute for a destination, by default, archiving online redo log files and standby redo log files is enabled at the destination, regardless of whether the database is running in the primary or the standby role. This default behavior is equivalent to setting the (ALL_LOGFILES, ALL_ROLES) keyword pair on the VALID_FOR attribute. For example:


```
LOG_ARCHIVE_DEST_1='LOCATION=/disk1/oracle/oradata/payroll/arch/
VALID_FOR=(ALL_LOGFILES,ALL_ROLES)
```
- The VALID_FOR attribute enables you to use the same initialization parameter file for both the primary and standby roles.

Example

The following example shows the default VALID_FOR keyword pair:

```
LOG_ARCHIVE_DEST_1='LOCATION=/disk1/oracle/oradata VALID_FOR=(ALL LOGFILES, ALL_ROLES)'
```

When this database is running in either the primary or standby role, destination 1 archives all log files to the /disk1/oracle/oradata local directory location.

See the scenarios in [Section 12.1](#) for detailed examples of various Data Guard configurations using the VALID_FOR attribute.

VERIFY

Indicates whether or not an archiver (ARC*n*) process should scan and verify the correctness of the contents of a completed archived redo log file, either local or remote, after successfully completing the archival operation.

Category	VERIFY
Data Type	Keyword
Valid values	Not applicable
Default Value	Not applicable
Requires attributes ...	None
Conflicts with attributes ...	LGWR
Corresponds to ...	VERIFY and ARCHIVER columns in the V\$ARCHIVE_DEST view

Usage Notes

- If the VERIFY attribute is not specified, archived redo log files will not be verified.
- The verification is significantly more thorough than the normal checksum verification that is always performed; the redo verification may take a substantial amount of time to complete.
- The use of the VERIFY attribute may have an affect on primary database performance.

Example

The following example shows the VERIFY attribute:

```
LOG_ARCHIVE_DEST_1='LOCATION=/disk1/oracle/oradata VERIFY'
LOG_ARCHIVE_DEST_2='LOCATION=/arch1/SRLs/ VALID_FOR=(STANDBY_LOGFILE, STANDBY_
ROLE) VERIFY'
LOG_ARCHIVE_DEST_3='SERVICE=denver VALID_FOR=(ONLINE_LOGFILE, PRIMARY_ROLE) VERIFY'
```

SQL Statements Relevant to Data Guard

This chapter summarizes the SQL and SQL*Plus statements that are useful for performing operations on standby databases in a Data Guard environment. This chapter includes the following topics:

- [ALTER DATABASE Statements](#)
- [ALTER SESSION Statements](#)

This chapter contains only the syntax and a brief summary of particular SQL statements. You must refer to the *Oracle Database SQL Reference* for complete syntax and descriptions about these and other SQL statements

See [Chapter 13](#) for a list of initialization parameters that you can set and dynamically update using the `ALTER SYSTEM SET` statement.

15.1 ALTER DATABASE Statements

[Table 15–1](#) describes `ALTER DATABASE` statements that are relevant to Data Guard.

Table 15–1 ALTER DATABASE Statements Used in Data Guard Environments

ALTER DATABASE Statement	Description
<pre>ADD [STANDBY] LOGFILE [THREAD <i>integer</i>] [GROUP <i>integer</i>] <i>filespec</i></pre>	<p>Adds one or more online redo log file groups or standby redo log file groups to the specified thread, making the log files available to the instance to which the thread is assigned.</p> <p>See Section 8.3.5 for an example of this statement.</p>
<pre>ADD [STANDBY] LOGFILE MEMBER '<i>filename</i>' [REUSE] TO <i>logfile-descriptor</i></pre>	<p>Adds new members to existing online redo log file groups or standby redo log file groups.</p> <p>See Section 5.7.3.2 for an example of this statement.</p>
<pre>[ADD DROP] SUPPLEMENTAL LOG DATA {PRIMARY KEY UNIQUE INDEX} COLUMNS</pre>	<p>This statement is for logical standby databases only.</p> <p>Use it to enable full supplemental logging before you create a logical standby database. This is necessary because supplemental logging is the source of change to a logical standby database. To implement full supplemental logging, you must specify either the <code>PRIMARY KEY COLUMNS</code> or the <code>UNIQUE INDEX COLUMNS</code> keyword on this statement.</p> <p>See <i>Oracle Database SQL Reference</i> for more information.</p>

Table 15–1 (Cont.) ALTER DATABASE Statements Used in Data Guard Environments

ALTER DATABASE Statement	Description
COMMIT TO SWITCHOVER TO [[PRIMARY] [[PHYSICAL LOGICAL] [STANDBY]] [WITH WITHOUT] SESSION SHUTDOWN] [WAIT NOWAIT]	Performs a switchover to: <ul style="list-style-type: none"> ■ Change the current primary database to the standby database role ■ Change one standby database to the primary database role. <p>Note: On logical standby databases, you issue the ALTER DATABASE PREPARE TO SWITCHOVER statement to prepare the database for the switchover before you issue the ALTER DATABASE COMMIT TO SWITCHOVER statement.</p> <p>See Section 7.2.1 and Section 7.3.1 for examples of this statement.</p>
CREATE [PHYSICAL LOGICAL] STANDBY CONTROLFILE AS ' <i>filename</i> ' [REUSE]	Creates a control file to be used to maintain a physical or a logical standby database. Issue this statement on the primary database. <p>See Section 3.2.2 for an example of this statement.</p>
DROP [STANDBY] LOGFILE <i>logfile_descriptor</i>	Drops all members of an online redo log file group or standby redo log file group. <p>See Section 8.3.5 for an example of this statement.</p>
DROP [STANDBY] LOGFILE MEMBER ' <i>filename</i> '	Drops one or more online redo log file members or standby redo log file members.
[NO]FORCE LOGGING	Controls whether or not the Oracle database logs all changes in the database except for changes to temporary tablespaces and temporary segments. The [NO]FORCE LOGGING clause is required to prevent inconsistent standby databases. <p>The primary database must be mounted but not open when you issue this statement. See Section 3.1.1 for an example of this statement.</p>
MOUNT [STANDBY DATABASE]	Mounts a standby database, allowing the standby instance to receive redo data from the primary instance.
OPEN	Opens a previously started and mounted database: <ul style="list-style-type: none"> ■ Physical standby databases are opened in read-only mode, restricting users to read-only transactions and preventing the generating of redo data. ■ Logical standby database are opened in read/write mode. <p>See Section x.x for an example of this statement.</p>
PREPARE TO SWITCHOVER TO [PRIMARY] [[PHYSICAL LOGICAL] [STANDBY]] [WITH WITHOUT] SESSION SHUTDOWN] [WAIT NOWAIT]	This statement is for logical standby databases only. <p>It prepares the primary database and the logical standby database for a switchover by building the LogMiner dictionary <i>before</i> the switchover takes place. After the dictionary build has completed, issue the ALTER DATABASE COMMIT TO SWITCHOVER statement to switch the roles of the primary and logical standby databases.</p> <p>See Section 7.3.1 for examples of this statements.</p>

Table 15–1 (Cont.) ALTER DATABASE Statements Used in Data Guard Environments

ALTER DATABASE Statement	Description
RECOVER MANAGED STANDBY DATABASE [[NODELAY] [DISCONNECT [FROM SESSION]] [NOPARALLEL PARALLEL [<i>integer</i>]] [UNTIL CHANGE <i>scn</i>] [USING CURRENT LOGFILE]]	<p>This statement starts and controls Redo Apply on physical standby databases. You can use the RECOVER MANAGED STANDBY DATABASE clause on a physical standby database that is mounted, open, or closed. See Step 2 in Section 3.2.6 and Section 6.3 for examples.</p> <p>Note: Several clauses and keywords were deprecated and are supported for backward compatibility only. See <i>Oracle Database SQL Reference</i> for more information about these clauses.</p>
RECOVER MANAGED STANDBY DATABASE CANCEL [[NOWAIT] [WAIT]] [IMMEDIATE]]	The CANCEL clause cancels Redo Apply on a physical standby database after applying the current archived redo log file.
RECOVER MANAGED STANDBY DATABASE FINISH [FORCE] [NOWAIT WAIT]]	<p>The FINISH clause initiates failover on the target physical standby database and recovers the current standby redo log files. Use the FINISH clause only in the event of the failure of the primary database. This clause overrides any delay intervals specified.</p> <p>Include FORCE to terminate the RFS processes and allow the failover to occur immediately, without waiting for the RFS process to exit. Specify NOWAIT to have control returned immediately, rather than after the recovery process is complete. See Step 4 in Section 7.2.2 for examples.</p>
REGISTER [OR REPLACE] [PHYSICAL LOGICAL] LOGFILE <i>filespec</i>	<p>Allows the registration of manually archived redo log files. See Section 5.8.4 for an example of this statement.</p>
RECOVER TO LOGICAL STANDBY <i>new_database_name</i>	Instructs log apply services to continue applying changes to the <i>physical</i> standby database until you issue the command to convert the database to a <i>logical</i> standby database. See Section 4.2.4.1 for more information.
RESET DATABASE TO INCARNATION <i>integer</i>	Resets the target recovery incarnation for the database from the current incarnation to a different incarnation.
SET STANDBY DATABASE TO MAXIMIZE {PROTECTION AVAILABILITY PERFORMANCE}	Use this clause to specify the level of protection for the data in your Data Guard configuration. You specify this clause from the primary database, which must be mounted but not open.

Table 15–1 (Cont.) ALTER DATABASE Statements Used in Data Guard Environments

ALTER DATABASE Statement	Description
START LOGICAL STANDBY APPLY INITIAL [<i>scn-value</i>] [[NEW PRIMARY <i>dblink</i>]	This statement is for logical standby databases only. It starts SQL Apply on a logical standby database. See Section 6.4.1 for examples of this statement.
{STOP ABORT} LOGICAL STANDBY APPLY	This statement is for logical standby databases only. Use the STOP clause to stop SQL Apply on a logical standby database in an orderly fashion. Use the ABORT clause to stop SQL Apply abruptly. See Section 7.3.2 for an example of this statement.
ACTIVATE [PHYSICAL LOGICAL] STANDBY DATABASE FINISH APPLY]	Performs a failover. The standby database must be mounted before it can be activated with this statement. Note: Do not use the ALTER DATABASE ACTIVATE STANDBY DATABASE statement to failover because it causes data loss. Instead, use the following best practices: <ul style="list-style-type: none"> For physical standby databases, use the ALTER DATABASE RECOVER MANAGED STANDBY DATABASE statement with the FINISH keyword to perform the role transition as quickly as possible with little or no data loss and without rendering other standby databases unusable. Note: The failover operation adds an end-of-redo marker to the header of the last log file being archived and sends the redo to all enabled destinations that are valid for the primary role (specified with the VALID_FOR= (PRIMARY_ROLE, *_LOGFILES) or the VALID_FOR= (ALL_ROLES, *_LOGFILES) attributes). For logical standby databases, use the ALTER DATABASE PREPARE TO SWITCHOVER and ALTER DATABASE COMMIT TO SWITCHOVER statements.

15.2 ALTER SESSION Statements

[Table 15–2](#) describes an ALTER SESSION statement that is relevant to Data Guard.

Table 15–2 ALTER SESSION Statement Used in Data Guard Environments

ALTER SESSION Statement	Description
ALTER SESSION [ENABLE DISABLE] GUARD	This statement is for logical standby databases only. This statement allows privileged users to turn the database guard on and off for the current session. See Section 7.3.2 for an example of this statement.

Views Relevant to Oracle Data Guard

This chapter describes the views that are significant in a Data Guard environment. The view described in this chapter are a subset of the views that are available for Oracle databases.

[Table 16–1](#) describes the views and indicates if a view applies to physical standby databases, logical standby databases, or primary databases. See *Oracle Database Reference* for complete information about views.

Table 16–1 Views That Are Pertinent to Data Guard Configurations

View	Database	Description
DBA_LOGSTDBY_EVENTS	Logical only	Contains information about the activity of a logical standby database. It can be used to determine the cause of failures that occur when SQL Apply is applying redo to a logical standby database.
DBA_LOGSTDBY_HISTORY	Logical only	Displays the history of switchovers and failovers for logical standby databases in a Data Guard configuration. It does this by showing the complete sequence of redo log streams processed or created on the local system, across all role transitions. (After a role transition, a new log stream is started and the log stream sequence number is incremented by the new primary database.)
DBA_LOGSTDBY_LOG	Logical only	Shows the log files registered for logical standby databases.
DBA_LOGSTDBY_NOT_UNIQUE	Logical only	Identifies tables that have no primary and no non-null unique indexes.
DBA_LOGSTDBY_PARAMETERS	Logical only	Contains the list of parameters used by SQL Apply.
DBA_LOGSTDBY_SKIP	Logical only	Lists the tables that will be skipped by SQL Apply.
DBA_LOGSTDBY_SKIP_TRANSACTION	Logical only	Lists the skip settings chosen.
DBA_LOGSTDBY_UNSUPPORTED	Logical only	Identifies the schemas and tables (and columns in those tables) that contain unsupported data types. Use this view when you are preparing to create a logical standby database.
V\$ARCHIVE_DEST	Primary, physical, and logical	Describes all of the destinations in the Data Guard configuration, including each destination's current value, mode, and status. Note: The information in this view does not persist across an instance shutdown.
V\$ARCHIVE_DEST_STATUS	Primary, physical, and logical	Displays runtime and configuration information for the archived redo log destinations. Note: The information in this view does not persist across an instance shutdown.

Table 16–1 (Cont.) Views That Are Pertinent to Data Guard Configurations

View	Database	Description
V\$ARCHIVE_GAP	Physical and logical	Displays information to help you identify a gap in the archived redo log files.
V\$ARCHIVED_LOG	Primary, physical, and logical	Displays archive redo log information from the control file, including names of the archived redo log files.
V\$DATABASE	Primary, physical, and logical	Provides database information from the control file. Includes information about fast-start failover (available only with the Data Guard broker).
V\$DATABASE_INCARNATION	Primary, physical, and logical	Displays information about all database incarnations. Oracle Database creates a new incarnation whenever a database is opened with the RESETLOGS option. Records about the current and the previous incarnation are also contained in the V\$DATABASE view.
V\$DATAFILE	Primary, physical, and logical	Provides datafile information from the control file.
V\$DATAGUARD_CONFIG	Primary, physical, and logical	Lists the unique database names defined with the DB_UNIQUE_NAME and LOG_ARCHIVE_CONFIG initialization parameters.
V\$DATAGUARD_STATS	Primary, physical, and logical	Displays how much redo data generated by the primary database is not yet available on the standby database, showing how much redo data could be lost if the primary database were to crash at the time you queried this view. You can query this view on any instance of a standby database in a Data Guard configuration. If you query this view on a primary database, then the column values are cleared.
V\$DATAGUARD_STATUS	Primary, physical, and logical	Displays and records events that would typically be triggered by any message to the alert log or server process trace files.
V\$LOG	Primary, physical, and logical	Contains log file information from the online redo log files.
V\$LOGFILE	Primary, physical, and logical	Contains information about the online redo log files and standby redo log files.
V\$LOG_HISTORY	Primary, physical, and logical	Contains log history information from the control file.
V\$LOGSTDBY_PROCESS	Logical only	Provides dynamic information about what is happening with SQL Apply. This view is very helpful when you are diagnosing performance problems during SQL Apply on the logical standby database, and it can be helpful for other problems.
V\$LOGSTDBY_PROGRESS	Logical only	Displays the progress of SQL Apply on the logical standby database.
V\$LOGSTDBY_STATE	Logical only	Consolidates information from the V\$LOGSTDBY_PROCESS and V\$LOGSTDBY_STATS views about the running state of SQL Apply and the logical standby database.
V\$LOGSTDBY_STATS	Logical only	Displays LogMiner statistics, current state, and status information for a logical standby database during SQL Apply. If SQL Apply is not running, the values for the statistics are cleared.

Table 16–1 (Cont.) Views That Are Pertinent to Data Guard Configurations

View	Database	Description
V\$LOGSTDBY_TRANSACTION	Logical only	Displays information about all active transactions being processed by SQL Apply on the logical standby database.
V\$MANAGED_STANDBY	Physical only	Displays current status information for Oracle database processes related to physical standby databases. Note: The information in this view does not persist across an instance shutdown.
V\$STANDBY_LOG	Physical and logical	Contains log file information from the standby redo log files.



Part III

Appendixes

This part contains the following appendixes:

- [Appendix A, "Troubleshooting Data Guard"](#)
- [Appendix B, "Upgrading Databases in a Data Guard Configuration"](#)
- [Appendix C, "Data Type and DDL Support on a Logical Standby Database"](#)
- [Appendix D, "Data Guard and Real Application Clusters"](#)
- [Appendix E, "Cascaded Destinations"](#)
- [Appendix F, "Creating a Standby Database with Recovery Manager"](#)
- [Appendix G, "Setting Archive Tracing"](#)

Troubleshooting Data Guard

This appendix provides help troubleshooting a standby database. This appendix contains the following sections:

- [Common Problems](#)
- [Log File Destination Failures](#)
- [Handling Logical Standby Database Failures](#)
- [Problems Switching Over to a Standby Database](#)
- [What to Do If SQL Apply Stops](#)
- [Network Tuning for Redo Data Transmission](#)
- [Slow Disk Performance on Standby Databases](#)
- [Log Files Must Match to Avoid Primary Database Shutdown](#)
- [Troubleshooting a Logical Standby Database](#)

A.1 Common Problems

If you encounter a problem when using a standby database, it is probably because of one of the following reasons:

- [Standby Archive Destination Is Not Defined Properly](#)
- [Renaming Datafiles with the ALTER DATABASE Statement](#)
- [Standby Database Does Not Receive Redo Data from the Primary Database](#)
- [You Cannot Mount the Physical Standby Database](#)

A.1.1 Standby Archive Destination Is Not Defined Properly

If the `STANDBY_ARCHIVE_DEST` initialization parameter does not specify a valid directory name on the standby database, the Oracle database will not be able to determine the directory in which to store the archived redo log files. Check the `DESTINATION` and `ERROR` columns in the `V$ARCHIVE_DEST` view by entering the following query and ensure the destination is valid:

```
SQL> SELECT DESTINATION, ERROR FROM V$ARCHIVE_DEST;
```

A.1.2 Renaming Datafiles with the ALTER DATABASE Statement

You cannot rename the datafile on the standby site when the `STANDBY_FILE_MANAGEMENT` initialization parameter is set to `AUTO`. When you set the `STANDBY_`

FILE_MANAGEMENT initialization parameter to AUTO, use of the following SQL statements is not allowed:

- ALTER DATABASE RENAME
- ALTER DATABASE ADD/DROP LOGFILE
- ALTER DATABASE ADD/DROP STANDBY LOGFILE MEMBER
- ALTER DATABASE CREATE DATAFILE AS

If you attempt to use any of these statements on the standby database, an error is returned. For example:

```
SQL> ALTER DATABASE RENAME FILE '/disk1/oracle/oradata/payroll/t_db2.log' to 'dummy';
alter database rename file '/disk1/oracle/oradata/payroll/t_db2.log' to 'dummy'
*
ERROR at line 1:
ORA-01511: error in renaming log/datafiles
ORA-01270: RENAME operation is not allowed if STANDBY_FILE_MANAGEMENT is auto
```

See [Section 8.3.1](#) to learn how to add datafiles to a physical standby database.

A.1.3 Standby Database Does Not Receive Redo Data from the Primary Database

If the standby site is not receiving redo data, query the V\$ARCHIVE_DEST view and check for error messages. For example, enter the following query:

```
SQL> SELECT DEST_ID "ID",
2> STATUS "DB_status",
3> DESTINATION "Archive_dest",
4> ERROR "Error"
5> FROM V$ARCHIVE_DEST WHERE DEST_ID <=5;
```

ID	DB_status	Archive_dest	Error
1	VALID	/vobs/oracle/work/arc_dest/arc	
2	ERROR	standby1	ORA-16012: Archivelog standby database identifier mismatch
3	INACTIVE		
4	INACTIVE		
5	INACTIVE		

5 rows selected.

If the output of the query does not help you, check the following list of possible issues. If any of the following conditions exist, redo transport services will fail to transmit redo data to the standby database:

- The service name for the standby instance is not configured correctly in the tnsnames.ora file for the primary database.
- The Oracle Net service name specified by the LOG_ARCHIVE_DEST_n parameter for the primary database is incorrect.
- The LOG_ARCHIVE_DEST_STATE_n parameter for the standby database is not set to the value ENABLE.
- The listener.ora file has not been configured correctly for the standby database.
- The listener is not started at the standby site.
- The standby instance is not started.
- You have added a standby archiving destination to the primary SPFILE or text initialization parameter file, but have not yet enabled the change.

- The databases in the Data Guard configuration are not all using a password file, or the SYS password contained in the password file is not identical on all systems.
- You used an invalid backup as the basis for the standby database (for example, you used a backup from the wrong database, or did not create the standby control file using the correct method).

A.1.4 You Cannot Mount the Physical Standby Database

You cannot mount the standby database if the standby control file was not created with the `ALTER DATABASE CREATE [LOGICAL] STANDBY CONTROLFILE . . .` statement or `RMAN` command. You cannot use the following types of control file backups:

- An operating system-created backup
- A backup created using an `ALTER DATABASE` statement *without* the `PHYSICAL STANDBY` or `LOGICAL STANDBY` option

A.2 Log File Destination Failures

If you specify `REOPEN` for an `OPTIONAL` destination, it is possible for the Oracle database to reuse online redo log files even if there is an error archiving to the destination in question. If you specify `REOPEN` for a `MANDATORY` destination, redo transport services stall the primary database when redo data cannot be successfully transmitted.

The `REOPEN` attribute is required when you use the `MAX_FAILURE` attribute.

[Example A–1](#) shows how to set a retry time of 5 seconds and limit retries to 3 times.

Example A–1 Setting a Retry Time and Limit

```
LOG_ARCHIVE_DEST_1='LOCATION=/arc_dest REOPEN=5 MAX_FAILURE=3'
```

Use the `ALTERNATE` attribute of the `LOG_ARCHIVE_DEST_n` parameter to specify alternate archive destinations. An alternate archiving destination can be used when the transmission of redo data to a standby database fails. If transmission fails and the `REOPEN` attribute was not specified or the `MAX_FAILURE` attribute threshold was exceeded, redo transport services attempts to transmit redo data to the alternate destination on the next archival operation.

Use the `NOALTERNATE` attribute to prevent the original archive destination from automatically changing to an alternate archive destination when the original archive destination fails.

[Example A–2](#) shows how to set the initialization parameters so that a single, mandatory, local destination will automatically fail over to a different destination if any error occurs.

Example A–2 Specifying an Alternate Destination

```
LOG_ARCHIVE_DEST_1='LOCATION=/disk1 MANDATORY ALTERNATE=LOG_ARCHIVE_DEST_2'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_2='LOCATION=/disk2 MANDATORY'
LOG_ARCHIVE_DEST_STATE_2=ALTERNATE
```

If the `LOG_ARCHIVE_DEST_1` destination fails, the archiving process will automatically switch to the `LOG_ARCHIVE_DEST_2` destination at the next log file switch on the primary database.

A.3 Handling Logical Standby Database Failures

An important tool for handling logical standby database failures is the `DBMS_LOGSTDBY.SKIP_ERROR` procedure. Depending on how important a table is, you might want to do one of the following:

- Ignore failures for a table or specific DDL
- Associate a stored procedure with a filter so at runtime a determination can be made about skipping the statement, executing this statement, or executing a replacement statement

Taking one of these actions prevents SQL Apply from stopping. Later, you can query the `DBA_LOGSTDBY_EVENTS` view to find and correct any problems that exist. See *Oracle Database PL/SQL Packages and Types Reference* for more information about using the `DBMS_LOGSTDBY` package with PL/SQL callout procedures.

A.4 Problems Switching Over to a Standby Database

In most cases, following the steps described in [Chapter 7](#) will result in a successful switchover. However, if the switchover is unsuccessful, the following sections may help you to resolve the problem:

- [Switchover Fails Because Redo Data Was Not Transmitted](#)
- [Switchover Fails Because SQL Sessions Are Still Active](#)
- [Switchover Fails Because User Sessions Are Still Active](#)
- [Switchover Fails with the ORA-01102 Error](#)
- [Redo Data Is Not Applied After Switchover](#)
- [Roll Back After Unsuccessful Switchover and Start Over](#)

A.4.1 Switchover Fails Because Redo Data Was Not Transmitted

If the switchover does not complete successfully, you can query the `SEQUENCE#` column in the `V$ARCHIVED_LOG` view to see if the last redo data transmitted from the original primary database was applied on the standby database. If the last redo data was not transmitted to the standby database, you can manually copy the archived redo log file containing the redo data from the original primary database to the old standby database and register it with the `SQL ALTER DATABASE REGISTER LOGFILE file_specification` statement. If you then start log apply services, the archived redo log file will be applied automatically. Query the `SWITCHOVER_STATUS` column in the `V$DATABASE` view. The `TO PRIMARY` value in the `SWITCHOVER_STATUS` column verifies switchover to the primary role is now possible.

```
SQL> SELECT SWITCHOVER_STATUS FROM V$DATABASE;
SWITCHOVER_STATUS
-----
TO PRIMARY
1 row selected
```

See [Chapter 16](#) for information about other valid values for the `SWITCHOVER_STATUS` column of the `V$DATABASE` view.

To continue with the switchover, follow the instructions in [Section 7.2.1](#) for physical standby databases or [Section 7.3.1](#) for logical standby databases, and try again to switch the target standby database to the primary role.

A.4.2 Switchover Fails Because SQL Sessions Are Still Active

If you do not include the `WITH SESSION SHUTDOWN` clause as a part of the `ALTER DATABASE COMMIT TO SWITCHOVER TO PHYSICAL STANDBY` statement, active SQL sessions might prevent a switchover from being processed. Active SQL sessions can include other Oracle Database processes.

When sessions are active, an attempt to switch over fails with the following error message:

```
SQL> ALTER DATABASE COMMIT TO SWITCHOVER TO PHYSICAL STANDBY;
ALTER DATABASE COMMIT TO SWITCHOVER TO PHYSICAL STANDBY *
ORA-01093: ALTER DATABASE CLOSE only permitted with no sessions connected
```

Action: Query the `V$SESSION` view to determine which processes are causing the error. For example:

```
SQL> SELECT SID, PROCESS, PROGRAM FROM V$SESSION
  2> WHERE TYPE = 'USER'
  3> AND SID <> (SELECT DISTINCT SID FROM V$MYSTAT);
SID          PROCESS          PROGRAM
-----
          7          3537  oracle@nhclone2 (CJQ0)
          10
          14
          16
          19
          21
6 rows selected.
```

In the previous example, the `JOB_QUEUE_PROCESSES` parameter corresponds to the `CJQ0` process entry. Because the job queue process is a user process, it is counted as a SQL session that prevents switchover from taking place. The entries with no process or program information are threads started by the job queue controller.

Verify the `JOB_QUEUE_PROCESSES` parameter is set using the following SQL statement:

```
SQL> SHOW PARAMETER JOB_QUEUE_PROCESSES;
NAME                                TYPE          VALUE
-----
job_queue_processes                  integer       5
```

Then, set the parameter to 0. For example:

```
SQL> ALTER SYSTEM SET JOB_QUEUE_PROCESSES=0;
Statement processed.
```

Because `JOB_QUEUE_PROCESSES` is a dynamic parameter, you can change the value and have the change take effect immediately without having to restart the instance. You can now retry the switchover procedure.

Do not modify the parameter in your initialization parameter file. After you shut down the instance and restart it after the switchover completes, the parameter will be reset to the original value. This applies to both primary and physical standby databases.

[Table A-1](#) summarizes the common processes that prevent switchover and what corrective action you need to take.

Table A-1 Common Processes That Prevent Switchover

Type of Process	Process Description	Corrective Action
CJQ0	Job Queue Scheduler Process	Change the <code>JOB_QUEUE_PROCESSES</code> dynamic parameter to the value 0. The change will take effect immediately without having to restart the instance.
QMN0	Advanced Queue Time Manager	Change the <code>AQ_TM_PROCESSES</code> dynamic parameter to the value 0. The change will take effect immediately without having to restart the instance.
DBSNMP	Oracle Enterprise Manager Management Agent	Issue the <code>emctl stop agent</code> command from the operating system prompt.

A.4.3 Switchover Fails Because User Sessions Are Still Active

If the switchover fails and returns the error `ORA-01093 "Alter database close only permitted with no sessions connected"` it is usually because the `ALTER DATABASE COMMIT TO SWITCHOVER` statement implicitly closed the database, and if there are any other user sessions connected to the database, the close fails.

If you receive this error, disconnect any user sessions that are still connected to the database. To do this, query the `V$SESSION` fixed view to see which sessions are still active as shown in the following example:

```
SQL> SELECT SID, PROCESS, PROGRAM FROM V$SESSION;
```

```

      SID  PROCESS          PROGRAM
-----
      1  26900      oracle@dbuser-sun (PMON)
      2  26902      oracle@dbuser-sun (DBW0)
      3  26904      oracle@dbuser-sun (LGWR)
      4  26906      oracle@dbuser-sun (CKPT)
      5  26908      oracle@dbuser-sun (SMON)
      6  26910      oracle@dbuser-sun (RECO)
      7  26912      oracle@dbuser-sun (ARC0)
      8  26897      sqlplus@dbuser-sun (TNS V1-V3)
     11  26917      sqlplus@dbuser-sun (TNS V1-V3)

```

9 rows selected.

In this example, the first seven sessions are all Oracle Database background processes. Among the two SQL*Plus sessions, one is the current SQL*Plus session issuing the query, and the other is an extra session that should be disconnected before you re-attempt the switchover.

A.4.4 Switchover Fails with the ORA-01102 Error

Suppose the standby database and the primary database reside on the same site. After both the `ALTER DATABASE COMMIT TO SWITCHOVER TO PHYSICAL STANDBY` and the `ALTER DATABASE COMMIT TO SWITCHOVER TO PRIMARY` statements are successfully executed, shut down and restart the physical standby database and the primary database.

Note: It is not necessary to shut down and restart the physical standby database if it has not been opened read-only since the instance was started.

However, the startup of the second database fails with ORA-01102 error "cannot mount database in EXCLUSIVE mode."

This could happen during the switchover if you did not set the `DB_UNIQUE_NAME` parameter in the initialization parameter file that is used by the standby database (that is, the original primary database). If the `DB_UNIQUE_NAME` parameter of the standby database is not set, the standby and the primary databases both use the same mount lock and cause the ORA-01102 error during the startup of the second database.

Action: Add `DB_UNIQUE_NAME=unique_database_name` to the initialization parameter file used by the standby database, and shut down and restart the standby and primary databases.

A.4.5 Redo Data Is Not Applied After Switchover

The archived redo log files are not applied to the new standby database after the switchover.

This might happen because some environment or initialization parameters were not properly set after the switchover.

Action:

- Check the `tnsnames.ora` file at the new primary site and the `listener.ora` file at the new standby site. There should be entries for a listener at the standby site and a corresponding service name at the primary site.
- Start the listener at the standby site if it has not been started.
- Check if the `LOG_ARCHIVE_DEST_n` initialization parameter was set to properly transmit redo data from the primary site to the standby site. For example, query the `V$ARCHIVE_DEST` fixed view at the primary site as follows:

```
SQL> SELECT DEST_ID, STATUS, DESTINATION FROM V$ARCHIVE_DEST;
```

If you do not see an entry corresponding to the standby site, you need to set `LOG_ARCHIVE_DEST_n` and `LOG_ARCHIVE_DEST_STATE_n` initialization parameters.

- Set the `STANDBY_ARCHIVE_DEST` and `LOG_ARCHIVE_FORMAT` initialization parameters correctly at the standby site so that the archived redo log files are applied to the desired location.
- At the standby site, set the `DB_FILE_NAME_CONVERT` and `LOG_FILE_NAME_CONVERT` initialization parameters. Set the `STANDBY_FILE_MANAGEMENT` initialization parameter to `AUTO` if you want the standby site to automatically add new datafiles that are created at the primary site.

A.4.6 Roll Back After Unsuccessful Switchover and Start Over

For physical standby databases in situations where an error occurred and it is not possible to continue with the switchover, it might still be possible to revert the new physical standby database back to the primary role by using the following steps:

1. Connect to the new standby database (old primary), and issue the following statement to convert it back to the primary role:

```
SQL> ALTER DATABASE COMMIT TO SWITCHOVER TO PRIMARY;
```

If this statement is successful, then shut down (if necessary) and restart the database. Once restarted, the database will be running in the primary database role, and you do not need to perform any more steps.

If this statement is unsuccessful, then continue with Step 3.

2. When the switchover to change the role from primary to physical standby was initiated, a trace file was written in the log directory. This trace file contains the SQL statements required to re-create the original primary control file. Locate the trace file and extract the SQL statements into a temporary file. Execute the temporary file from SQL*Plus. This will revert the new standby database back to the primary role.
3. Shut down the original physical standby database.
4. Create a new standby control file. This is necessary to resynchronize the primary database and physical standby database. Copy the physical standby control file to the original physical standby system. [Section 3.2.2](#) describes how to create a physical standby control file.
5. Restart the original physical standby instance.

If this procedure is successful and archive gap management is enabled, the FAL processes will start and re-archive any missing archived redo log files to the physical standby database. Force a log switch on the primary database and examine the alert logs on both the primary database and physical standby database to ensure the archived redo log file sequence numbers are correct.

See [Section 5.8](#) for information about archive gap management and [Appendix G](#) for information about locating the trace files.

6. Try the switchover again.

At this point, the Data Guard configuration has been rolled back to its initial state, and you can try the switchover operation again (after correcting any problems that might have led to the initial unsuccessful switchover).

A.5 What to Do If SQL Apply Stops

Log apply services cannot apply unsupported DML statements, DDL statements, and Oracle supplied packages to a logical standby database running SQL Apply.

When an unsupported statement or package is encountered, SQL Apply stops. You can take the actions described in [Table A-2](#) to correct the situation and start SQL Apply on the logical standby database again.

Table A-2 Fixing Typical SQL Apply Errors

If...	Then...
You suspect an unsupported statement or Oracle supplied package was encountered	Find the last statement in the <code>DBA_LOGSTDBY_EVENTS</code> view. This will indicate the statement and error that caused SQL Apply to fail. If an incorrect SQL statement caused SQL Apply to fail, transaction information, as well as the statement and error information, can be viewed. The transaction information can be used with LogMiner tools to understand the cause of the problem.
An error requiring database management occurred, such as running out of space in a particular tablespace	Fix the problem and resume SQL Apply using the <code>ALTER DATABASE START LOGICAL STANDBY APPLY</code> statement.
An error occurred because a SQL statement was entered incorrectly, such as an incorrect standby database filename being entered in a tablespace statement	Enter the correct SQL statement and use the <code>DBMS_LOGSTDBY.SKIP_TRANSACTION</code> procedure to ensure the incorrect statement is ignored the next time SQL Apply is run. Then, restart SQL Apply using the <code>ALTER DATABASE START LOGICAL STANDBY APPLY</code> statement.
An error occurred because skip parameters were incorrectly set up, such as specifying that all DML for a given table be skipped but <code>CREATE</code> , <code>ALTER</code> , and <code>DROP TABLE</code> statements were not specified to be skipped	Issue the <code>DBMS_LOGSTDBY.SKIP('TABLE', 'schema_name', 'table_name', null)</code> procedure, then restart SQL Apply.

See [Chapter 16](#) for information about querying the `DBA_LOGSTDBY_EVENTS` view to determine the cause of failures.

A.6 Network Tuning for Redo Data Transmission

For optimal performance, set the Oracle Net SDU parameter to 32 kilobytes in each Oracle Net connect descriptor used by redo transport services.

The following example shows a database initialization parameter file segment that defines a remote destination `netserv`:

```
LOG_ARCHIVE_DEST_3='SERVICE=netserv'
```

The following example shows the definition of that service name in the `tnsnames.ora` file:

```
netserv=(DESCRIPTION=(SDU=32768)(ADDRESS=(PROTOCOL=tcp)(HOST=host)(PORT=1521))
(CONNECT_DATA=(SERVICE_NAME=srvc)))
```

The following example shows the definition in the `listener.ora` file:

```
LISTENER=(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=tcp)
(HOST=host)(PORT=1521))))
```

```
SID_LIST_LISTENER=(SID_LIST=(SID_DESC=(SDU=32768)(SID_NAME=sid)
(GLOBALDBNAME=srvc)(ORACLE_HOME=/oracle)))
```

If you archive to a remote site using a high-latency or high-bandwidth network link, you can improve performance by using the `SQLNET.SEND_BUF_SIZE` and `SQLNET.RECV_BUF_SIZE` Oracle Net profile parameters to increase the size of the network send and receive I/O buffers.

See *Oracle Database Net Services Administrator's Guide*.

A.7 Slow Disk Performance on Standby Databases

If asynchronous I/O on the file system itself is showing performance problems, try mounting the file system using the Direct I/O option or setting the `FILESYSTEMIO_OPTIONS=SETALL` initialization parameter. The maximum I/O size setting is 1 MB.

A.8 Log Files Must Match to Avoid Primary Database Shutdown

If you have configured a standby redo log on one or more standby databases in the configuration, ensure the size of the standby redo log files on each standby database exactly matches the size of the online redo log files on the primary database.

At log switch time, if there are no available standby redo log files that match the size of the new current online redo log file on the primary database:

- The primary database will shut down if it is operating in maximum protection mode,
- or*
- The RFS process on the standby database will create an archived redo log file on the standby database and write the following message in the alert log:

No standby log files of size <#> blocks available.

For example, if the primary database uses two online redo log groups whose log files are 100K, then the standby database should have 3 standby redo log groups with log file sizes of 100K.

Also, whenever you add a redo log group to the primary database, you must add a corresponding standby redo log group to the standby database. This reduces the probability that the primary database will be adversely affected because a standby redo log file of the required size is not available at log switch time.

See [Section 3.1.3, "Configure a Standby Redo Log"](#) and [Section 5.7, "Managing Log Files"](#) for more information.

A.9 Troubleshooting a Logical Standby Database

This section contains the following topics:

- [Recovering from Errors](#)
- [Troubleshooting SQL*Loader Sessions](#)
- [Troubleshooting Long-Running Transactions](#)
- [Troubleshooting ORA-1403 Errors with Flashback Transactions](#)

A.9.1 Recovering from Errors

Logical standby databases maintain user tables, sequences, and jobs. To maintain other objects, you must reissue the DDL statements seen in the redo data stream.

If SQL Apply fails, an error is recorded in the `DBA_LOGSTDBY_EVENTS` table. The following sections demonstrate how to recover from two such errors.

A.9.1.1 DDL Transactions Containing File Specifications

DDL statements are executed the same way on the primary database and the logical standby database. If the underlying file structure is the same on both databases, the DDL will execute on the standby database as expected.

If an error was caused by a DDL transaction containing a file specification that did not match in the logical standby database environment, perform the following steps to fix the problem:

1. Use the `ALTER SESSION DISABLE GUARD` statement to bypass the database guard so you can make modifications to the logical standby database:

```
SQL> ALTER SESSION DISABLE GUARD;
```

2. Execute the DDL statement, using the correct file specification, and then reenble the database guard. For example:

```
SQL> ALTER TABLESPACE t_table ADD DATAFILE '/dbs/t_db.f' SIZE 100M REUSE;
SQL> ALTER SESSION ENABLE GUARD;
```

3. Start SQL Apply on the logical standby database and skip the failed transaction.

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE
2> SKIP FAILED TRANSACTION;
```

In some situations, the problem that caused the transaction to fail can be corrected and SQL Apply restarted without skipping the transaction. An example of this might be when available space is exhausted. (Do not let the primary and logical standby databases diverge when skipping DDL transactions. If possible, you should manually execute a compensating transaction in place of the skipped transaction.)

The following example shows SQL Apply stopping, the error being corrected, and then restarting SQL Apply:

```
SQL> SET LONG 1000
SQL> ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-YY HH24:MI:SS';
```

Session altered.

```
SQL> SELECT EVENT_TIME, COMMIT_SCN, EVENT, STATUS FROM DBA_LOGSTDBY_EVENTS;
```

```
EVENT_TIME          COMMIT_SCN
-----
EVENT
-----
STATUS
-----
22-OCT-03 15:47:58
```

ORA-16111: log mining and apply setting up

```
22-OCT-03 15:48:04          209627
insert into "SCOTT"."EMP"
values
  "EMPNO" = 7900,
  "ENAME" = 'ADAMS',
  "JOB" = 'CLERK',
  "MGR" IS NULL,
  "HIREDATE" = TO_DATE('22-OCT-03', 'DD-MON-RR'),
  "SAL" = 950,
  "COMM" IS NULL,
```

```
"DEPTNO" IS NULL
ORA-01653: unable to extend table SCOTT.EMP by %200 bytes in tablespace T_TABLE
```

In the example, the ORA-01653 message indicates that the tablespace was full and unable to extend itself. To correct the problem, add a new datafile to the tablespace. For example:

```
SQL> ALTER TABLESPACE t_table ADD DATAFILE '/dbs/t_db.f' SIZE 60M;
Tablespace altered.
```

Then, restart SQL Apply:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE;
Database altered.
```

When SQL Apply restarts, the transaction that failed will be reexecuted and applied to the logical standby database.

A.9.1.2 Recovering from DML Failures

Do not use the `SKIP_TRANSACTION` procedure to filter DML failures. Not only is the DML that is seen in the events table skipped, but so is all the DML associated with the transaction. This will cause multiple tables.

DML failures usually indicate a problem with a specific table. For example, assume the failure is an out-of-storage error that you cannot resolve immediately. The following steps demonstrate one way to respond to this problem.

1. Bypass the table, but not the transaction, by adding the table to the skip list:

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP('DML', 'SCOTT', 'EMP');
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE;
```

From this point on, DML activity for the `SCOTT.EMP` table is not applied. After you correct the storage problem, you can fix the table, provided you set up a database link to the primary database that has administrator privileges to run procedures in the `DBMS_LOGSTDBY` package.

2. Using the database link to the primary database, drop the local `SCOTT.EMP` table and then re-create it, and pull the data over to the standby database.

```
SQL> ALTER DATABASE STOP LOGICAL STANDBY APPLY;
SQL> EXECUTE DBMS_LOGSTDBY.INSTANTIATE_TABLE('SCOTT', 'EMP', 'PRIMARYDB');
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE;
```

3. To ensure a consistent view across the newly instantiated table and the rest of the database, wait for SQL Apply to catch up with the primary database before querying this table. Refer to [Section 9.4.6, "Adding or Re-Creating Tables On a Logical Standby Database"](#) for a detailed example.

A.9.2 Troubleshooting SQL*Loader Sessions

Oracle SQL*Loader provides a method of loading data from different sources into the Oracle Database. This section analyzes some of the features of the SQL*Loader utility as it pertains to SQL Apply.

Regardless of the method of data load chosen, the SQL*Loader control files contain an instruction on what to do to the current contents of the Oracle table into which the new data is to be loaded, via the keywords of `APPEND` and `REPLACE`. The following examples show how to use these keywords on a table named `LOAD_STOK`:

- When using the `APPEND` keyword, the new data to be loaded is appended to the contents of the `LOAD_STOK` table:

```
LOAD DATA
INTO TABLE LOAD_STOK APPEND
```

- When using the `REPLACE` keyword, the contents of the `LOAD_STOK` table are deleted prior to loading new data. Oracle SQL*Loader uses the `DELETE` statement to purge the contents of the table, in a single transaction:

```
LOAD DATA
INTO TABLE LOAD_STOK REPLACE
```

Rather than using the `REPLACE` keyword in the SQL*Loader script, Oracle recommends that prior to loading the data, issue the SQL*Plus `TRUNCATE TABLE` command against the table on the primary database. This will have the same effect of purging both the primary and standby databases copy of the table in a manner that is both fast and efficient because the `TRUNCATE TABLE` command is recorded in the online redo log files and is issued by SQL Apply on the logical standby database.

The SQL*Loader script may continue to contain the `REPLACE` keyword, but it will now attempt to `DELETE` zero rows from the object on the primary database. Because no rows were deleted from the primary database, there will be no redo recorded in the redo log files. Therefore, no `DELETE` statement will be issued against the logical standby database.

Issuing the `REPLACE` keyword without the DDL command `TRUNCATE TABLE` provides the following potential problems for SQL Apply when the transaction needs to be applied to the logical standby database.

- If the table currently contains a significant number of rows, then these rows need to be deleted from the standby database. Because SQL Apply is not able to determine the original syntax of the statement, SQL Apply must issue a `DELETE` statement for each row purged from the primary database.

For example, if the table on the primary database originally had 10,000 rows, then Oracle SQL*Loader will issue a single `DELETE` statement to purge the 10,000 rows. On the standby database, SQL Apply does not know that all rows are to be purged, and instead must issue 10,000 individual `DELETE` statements, with each statement purging a single row.

- If the table on the standby database does not contain an index that can be used by SQL Apply, then the `DELETE` statement will issue a Full Table Scan to purge the information.

Continuing with the previous example, because SQL Apply has issued 10,000 individual `DELETE` statements, this could result in 10,000 Full Table Scans being issued against the standby database.

A.9.3 Troubleshooting Long-Running Transactions

One of the primary causes for long-running transactions in a SQL Apply environment is because of Full Table Scans. Additionally, long-running transactions could be the result of DDL operations being replicated to the standby database, such as when creating or rebuilding an index.

Identifying Long-Running Transactions

If SQL Apply is executing a single SQL statement for a long period of time, then a warning message similar to the following is reported in the alert log of the SQL Apply instance:

```
Mon Feb 17 14:40:15 2003
WARNING: the following transaction makes no progress
WARNING: in the last 30 seconds for the given message!
WARNING: xid =
0x0016.007.000017b6 cscn = 1550349, message# = 28, slavid = 1
knacrb: no offending session found (not ITL pressure)
```

Note the following about the warning message:

- This warning is similar to the warning message returned for interested transaction list (ITL) pressure, with the exception being the last line that begins with `knacrb`. The final line indicates:
 - A Full Table Scan may be occurring
 - This issue has nothing to do with interested transaction list (ITL) pressure
- This warning message is reported only if a single statement takes more than 30 seconds to execute.

It may not be possible to determine the SQL statement being executed by the long-running statement, but the following SQL statement may help in identifying the database objects on which SQL Apply is operating:

```
SQL> SELECT SAS.SERVER_ID
2      , SS.OWNER
3      , SS.OBJECT_NAME
4      , SS.STATISTIC_NAME
5      , SS.VALUE
6 FROM V$SEGMENT_STATISTICS SS
7      , V$LOCK L
8      , V$STREAMS_APPLY_SERVER SAS
9 WHERE SAS.SERVER_ID = &SLAVE_ID
10      AND L.SID = SAS.SID
11      AND L.TYPE = 'TM'
12      AND SS.OBJ# = L.ID1;
```

Additionally, you can issue the following SQL statement to identify the SQL statement that has resulted in a large number of disk reads being issued per execution:

```
SQL> SELECT SUBSTR(SQL_TEXT,1,40)
2      , DISK_READS
3      , EXECUTIONS
4      , DISK_READS/EXECUTIONS
5      , HASH_VALUE
6      , ADDRESS
7 FROM V$SQLAREA
8 WHERE DISK_READS/GREATEST(EXECUTIONS,1) > 1
9      AND ROWNUM < 10
10 ORDER BY DISK_READS/GREATEST(EXECUTIONS,1) DESC
```

Oracle recommends that all tables have primary key constraints defined, which automatically means that the column is defined as `NOT NULL`. For any table where a primary-key constraint cannot be defined, an index should be defined on an appropriate column that is defined as `NOT NULL`. If a suitable column does not exist on the table, then the table should be reviewed and, if possible, skipped by SQL Apply.

The following steps describe how to skip all DML statements issued against the FTS table on the SCOTT schema:

1. Stop SQL Apply:

```
SQL> ALTER DATABASE STOP LOGICAL STANDBY APPLY;
Database altered
```

2. Configure the skip procedure for the SCOTT.FTS table for all DML transactions:

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP(stmt => 'DML' , -
      schema_name => 'SCOTT' , -
      object_name => 'FTS');
PL/SQL procedure successfully completed
```

3. Start SQL Apply:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE;
Database altered
```

Troubleshooting ITL Pressure

Interested transaction list (ITL) pressure is reported in the alert log of the SQL Apply instance. [Example A-3](#) shows an example of the warning messages.

Example A-3 Warning Messages Reported for ITL Pressure

```
Tue Apr 22 15:50:42 2003
WARNING: the following transaction makes no progress
WARNING: in the last 30 seconds for the given message!
WARNING: xid =
0x0006.005.000029fa cscn = 2152982, message# = 2, slavid = 17
```

Real-Time Analysis

The messages shown in [Example A-3](#) indicate that the SQL Apply process (slavid) #17 has not made any progress in the last 30 seconds. To determine the SQL statement being issued by the Apply process, issue the following query:

```
SQL> SELECT SA.SQL_TEXT
2 FROM V$SQLAREA SA
3 , V$SESSION S
4 , V$STREAMS_APPLY_SERVER SAS
5 WHERE SAS.SERVER_ID = &SLAVEID
6 AND S.SID = SAS.SID
7 AND SA.ADDRESS = S.SQL_ADDRESS
SQL_TEXT
-----
insert into "APP"."LOAD_TAB_1" p("PK","TEXT")values(:1,:2)
```

An alternative method to identifying ITL pressure is to query the V\$LOCK view, as shown in the following example. Any session that has a request value of 4 on a TX lock, is waiting for an ITL to become available.

```
SQL> SELECT SID,TYPE, ID1, ID2, LMODE, REQUEST
2 FROM V$LOCK
3 WHERE TYPE = 'TX'
```

SID	TY	ID1	ID2	LMODE	REQUEST
8	TX	327688	48	6	0
10	TX	327688	48	0	4

In this example, SID 10 is waiting for the TX lock held by SID 8.

Post-Incident Review

Pressure for a segment's ITL is unlikely to last for an extended period of time. In addition, ITL pressure that lasts for less than 30 seconds will not be reported in the standby databases alert log. Therefore, to determine which objects have been subjected to ITL pressure, issue the following statement:

```
SQL> SELECT SEGMENT_OWNER, SEGMENT_NAME, SEGMENT_TYPE
2     FROM V$SEGMENT_STATISTICS
3     WHERE STATISTIC_NAME = 'ITL WAITS'
4     AND VALUE > 0
5     ORDER BY VALUE
```

This statement reports all database segments that have had ITL pressure at some time since the instance was last started.

Note: This SQL statement is not limited to a logical standby databases in the Data Guard environment. It is applicable to any Oracle database.

Resolving ITL Pressure

To increase the INITRANS integer for a particular database object, it is necessary to first stop SQL Apply.

See Also: *Oracle Database SQL Reference* for more information about specifying the INITRANS integer, which is the initial number of concurrent transaction entries allocated within each data block allocated to the database object

The following example shows the necessary steps to increase the INITRANS for table `load_tab_1` in the schema `app`.

1. Stop SQL Apply:

```
SQL> ALTER DATABASE STOP LOGICAL STANDBY APPLY;
Database altered.
```

2. Temporarily bypass the database guard:

```
SQL> ALTER SESSION DISABLE GUARD;
Session altered.
```

3. Increase the INITRANS on the standby database. For example:

```
SQL> ALTER TABLE APP.LOAD_TAB_1 INITRANS 30;
Table altered
```

4. Reenable the database guard:

```
SQL> ALTER SESSION ENABLE GUARD;
Session altered
```

5. Start SQL Apply:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE;
Database altered.
```

Also, consider modifying the database object on the primary database, so in the event of a switchover, the error should not occur on the new standby database.

A.9.4 Troubleshooting ORA-1403 Errors with Flashback Transactions

If SQL Apply returns the ORA-1403: No Data Found error, then it may be possible to use Flashback Transaction to reconstruct the missing data. This is reliant upon the UNDO_RETENTION initialization parameter specified on the standby database instance.

Under normal circumstances, the ORA-1403 error should not be seen in a logical standby database environment. The error occurs when data in a table that is being managed by SQL Apply is modified directly on the standby database and then the same data is modified on the primary database.

When the modified data is updated on the primary database and is subsequently received on the logical standby database, SQL Apply verifies the original version of the data is present on the standby database before updating the record. When this verification fails, the ORA-1403: No Data Found error is returned.

The Initial Error

When SQL Apply verification fails, the error message is reported in the alert log of the logical standby database and a record is inserted in the DBA_LOGSTDBY_EVENTS view.

The information in the alert log is truncated, while the error is reported in it's entirety in the database view. For example:

```
LOGSTDBY stmt: UPDATE "SCOTT"."MASTER"
SET
  "NAME" = 'john'
WHERE
  "PK" = 1 and
  "NAME" = 'andrew' and
  ROWID = 'AAAAAAAAEAAAAAPAAA'
LOGSTDBY status: ORA-01403: no data found
LOGSTDBY PID 1006, oracle@staco03 (P004)
LOGSTDBY XID 0x0006.00e.00000417, Thread 1, RBA 0x02dd.00002221.10
```

The Investigation

The first step is to analyze the historical data of the table that caused the error. This can be achieved using the VERSIONS clause of the SELECT statement. For example, you can issue the following query on the primary database:

```
SELECT VERSIONS_XID
       , VERSIONS_STARTSCN
       , VERSIONS_ENDSCN
       , VERSIONS_OPERATION
       , PK
       , NAME
FROM SCOTT.MASTER
     VERSIONS BETWEEN SCN MINVALUE AND MAXVALUE
WHERE PK = 1
ORDER BY NVL(VERSIONS_STARTSCN,0);
```

VERSIONS_XID	VERSIONS_STARTSCN	VERSIONS_ENDSCN	V	PK	NAME
03001900EE070000	3492279	3492290	I	1	andrew
02000D00E4070000	3492290		D	1	andrew

Depending upon the amount of undo retention that the database is configured to retain (UNDO_RETENTION) and the activity on the table, the information returned might be extensive and you may need to change the versions between syntax to restrict the amount of information returned.

From the information returned, it can be seen that the record was first inserted at SCN 3492279 and then was deleted at SCN 3492290 as part of transaction ID 02000D00E4070000.

Using the transaction ID, the database should be queried to find the scope of the transaction. This is achieved by querying the FLASHBACK_TRANSACTION_QUERY view.

```
SELECT OPERATION
       , UNDO_SQL
   FROM FLASHBACK_TRANSACTION_QUERY
  WHERE XID = HEXTORAW('02000D00E4070000');
```

OPERATION	UNDO_SQL
DELETE	insert into "SCOTT"."MASTER"("PK","NAME") values ('1','andrew');

```
BEGIN
```

Note that there is always one row returned representing the start of the transaction. In this transaction, only one row was deleted in the master table. The UNDO_SQL column when executed will restore the original data into the table.

```
SQL> INSERT INTO "SCOTT"."MASTER"("PK","NAME") VALUES ('1','ANDREW');
SQL> COMMIT;
```

When you restart SQL Apply, the transaction will be applied to the standby database:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE;
```

Upgrading Databases in a Data Guard Configuration

The procedures in this appendix describe how to upgrade to Oracle Database 10g Release 2 (10.2) when a physical or logical standby database is present in the configuration.

Note: This appendix describes the traditional method for upgrading your Oracle Database software with a logical standby database in place. A second method in [Chapter 11, "Using SQL Apply to Upgrade the Oracle Database"](#) describes how to upgrade with a logical standby database in place in a rolling fashion to minimize downtime. Use the steps from only one method to perform the complete upgrade. Do not attempt to use both methods or to combine the steps from the two methods as you perform the upgrade process.

This appendix contains the following topics:

- [Before You Upgrade the Oracle Database Software](#)
- [Upgrading Oracle Database with a Physical Standby Database In Place](#)
- [Upgrading Oracle Database with a Logical Standby Database In Place](#)

B.1 Before You Upgrade the Oracle Database Software

Consider the following points before beginning to upgrade your Oracle Database software:

- If you are using the Data Guard broker to manage your configuration, follow the instructions in *Oracle Data Guard Broker* manual for information about removing or disabling the broker configuration.
- The procedures in this appendix are to be used in conjunction with the ones contained in the *Oracle Database Upgrade Guide* for 10g release 2 (10.2).
- The procedures in this appendix use the Database Upgrade Assistant (DBUA) to perform the upgrade. For instructions on performing the upgrade manually, refer to the *Oracle Database Upgrade Guide*. The manual upgrade steps described should be performed whenever use of DBUA is mentioned.
- Check for nologging operations. If nologging operations have been performed then you must update the standby database. See [Section 12.10, "Recovering After the NOLOGGING Clause Is Specified"](#) for details.

- Make note of any tablespaces or datafiles that need recovery due to OFFLINE IMMEDIATE. Tablespaces or datafiles should be recovered and either online or offline prior to upgrading.

B.2 Upgrading Oracle Database with a Physical Standby Database In Place

Perform the following steps to upgrade to Oracle Database 10g Release 2 (10.2) when a physical standby database is present in the configuration:

1. Review and perform the steps listed in "Chapter 2 Preparing to Upgrade" of the *Oracle Database Upgrade Guide*.
2. On the primary and standby systems, log in to each system as the owner of the Oracle software directory, and set the environment to the existing (9.2 or 10.1) installation.
3. On the primary database, stop all user activity on the primary database.
4. If you are using Real Application Clusters, issue a SHUTDOWN NORMAL or SHUTDOWN IMMEDIATE statement on all but one primary database instance.

```
SQL> SHUTDOWN IMMEDIATE;
```

Then, on the remaining active database instance, archive the current log file. For example:

```
SQL> ALTER SYSTEM ARCHIVE LOG CURRENT;
```

This ensures all available archived redo data from the Real Application Clusters instances is transmitted to the standby database.

5. On the active primary database instance, identify and record the current log thread and sequence number. Then, archive the current log:

```
SQL> SELECT THREAD#, SEQUENCE# FROM V$LOG WHERE STATUS='CURRENT';
SQL> ALTER SYSTEM ARCHIVE LOG CURRENT;
```

6. Shut down the active primary database instance with NORMAL or IMMEDIATE priority. Stop all listeners, agents, and other processes running against the ORACLE_HOME.

```
SQL> SHUTDOWN IMMEDIATE;
% agentctl stop
% lsnrctl stop
```

7. On the standby system, if you are using Real Application Clusters, shut down (NORMAL or IMMEDIATE) all but one standby database instance. Place the remaining standby database instance is currently in managed recovery.
8. On the active standby instance that is running Redo Apply, query the V\$LOG_HISTORY view to verify that each log file archived in Step 5 has been received and applied to the standby database. For example:

```
SQL> SELECT MAX(SEQUENCE#) FROM V$LOG_HISTORY;
```

9. Once the last log has been applied stop Redo Apply, shut down the standby database, and stop all listeners and agents running against the current (9.2 or 10.1) database.

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE CANCEL;
```

```
SQL> SHUTDOWN IMMEDIATE;
% agentctl stop
% lsnrctl stop
```

10. On the standby system, install Oracle Database 10g Release 2 (10.2) into its own Oracle home using the Oracle Universal Installer, as described in the *Oracle Database Upgrade Guide*. To ensure an error-free upgrade, it is recommended that you also install the Companion CD. Do not perform any other upgrade steps.
11. Copy the server parameter file (SPFILE), password file, and any necessary networking files from the old (9.2 or 10.1) ORACLE_HOME directory into the new 10.2 ORACLE_HOME directory. Note that for the standby database to receive redo from the primary database, the standby database must have the REMOTE_LOGIN_EXCLUSIVE initialization parameter set to SHARED or EXCLUSIVE, a password file must exist, and the SYS password on the standby database must be the same as the SYS password on the primary database.
12. On the primary system, install Oracle Database 10g Release 2 (10.2) into its own Oracle home using the Oracle Universal Installer as described in the *Oracle Database Upgrade Guide*. To ensure an error-free upgrade, it is recommended that you also install the Companion CD.
13. Include the Oracle Net service name in the TNSNAMES.ORA file that resides in the release 10.2 Oracle_Home.
14. After 10.2 has been installed, with your environment still set to the old (9.2 or 10.1) installation, startup the primary database in UPGRADE mode:

```
SQL> STARTUP UPGRADE;
```

15. From the 10.2 Oracle_Home, start the Database Upgrade Assistant and upgrade the primary database.

```
% cd /u01/app/oracle/product/10.2/bin
% ./dbua
```

Note: The old (9.2 or 10.1) database must be included in the oratab file to be seen by the Database Upgrade Assistant. For complete information on using DBUA, see the *Oracle Database Upgrade Guide*.

Note: You may receive errors in the alert log on the primary database stating that the primary database is unable to contact the standby database. You can ignore this error; This error is expected because the standby database has not been restarted up to this point.

16. After the upgrade process has begun on the primary database, start the standby listener on the standby database running the new Oracle Database 10g Release 2 (10.2) software.
17. With the environment set to the new 10.2 software installation, issue the STARTUP NOMOUNT statement on the standby database. Ensure that the STANDBY_FILE_MANAGEMENT initialization parameter on the standby database is set to AUTO and that FAL_SERVER and FAL_CLIENT are properly configured:
 - The FAL_SERVER should be set to an Oracle Net service name that exists in the standby database's TNSNAMES.ORA file.

- The `FAL_CLIENT` should be set to an Oracle Net service name that exists in the primary database's `TNSNAMES.ORA` file that points to the standby database listener.

The Oracle Net service names must be able to be resolved in the new 10.2 `Oracle_Home`. For example:

```
SQL> STARTUP NOMOUNT
SQL> ALTER SYSTEM SET STANDBY_FILE_MANAGEMENT=AUTO SCOPE=BOTH;
SQL> ALTER SYSTEM SET FAL_SERVER=MTS SCOPE=BOTH;
SQL> ALTER SYSTEM SET FAL_CLIENT=MTS_PHY SCOPE=BOTH;
```

18. Mount the standby database and start Redo Apply:

```
SQL> ALTER DATABASE MOUNT STANDBY DATABASE;
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE DISCONNECT;
```

19. Once the Database Upgrade Assistant has completed, on the primary system, configure your environment to the new 10.2 `Oracle_Home` and connect to the primary database. Identify and record the current log thread and sequence number. Then, archive the current log:

```
SQL> SELECT THREAD#, SEQUENCE# FROM V$LOG WHERE STATUS='CURRENT';
SQL> ALTER SYSTEM ARCHIVE LOG CURRENT;
```

20. On the standby instance, query the `V$LOG_HISTORY` view to verify that each log file archived in Step 19 has been received and applied on the standby database:

```
SQL> SELECT MAX(SEQUENCE#) FROM V$LOG_HISTORY;
```

B.3 Upgrading Oracle Database with a Logical Standby Database In Place

Perform the following steps to upgrade Oracle Database 10g Release 2 (10.2) when a logical standby database is present in the configuration:

1. Review and perform the steps listed in "Chapter 2 Preparing to Upgrade" of the *Oracle Database Upgrade Guide*.
2. On the primary and standby systems, log in to each system as the owner of the Oracle software directory, and set the environment to the existing (9.2 or 10.1) installation.
3. On the primary system, stop all user activity on the primary database.
4. If you are using Real Application Clusters, issue a `SHUTDOWN NORMAL` or `SHUTDOWN IMMEDIATE` statement on all but one primary database instance.

```
SQL> SHUTDOWN IMMEDIATE;
```

Then, on the remaining active database instance, archive the current log file:

```
SQL> SHUTDOWN IMMEDIATE;
SQL> ALTER SYSTEM ARCHIVE LOG CURRENT;
```

This ensures all available archived redo data from the Real Application Clusters instances is transmitted to the standby database.

5. On the active primary database instance, identify and record the current log thread and sequence number. Then, archive the current log:

```
SQL> SELECT THREAD#, SEQUENCE# FROM V$LOG WHERE STATUS='CURRENT';
SQL> ALTER SYSTEM ARCHIVE LOG CURRENT;
```

6. If you are using Real Application Clusters on the standby system, issue the `SHUTDOWN NORMAL` or `SHUTDOWN IMMEDIATE` statement on all but one standby instance.
7. On the active standby instance, verify that each log file archived in Step 5 has been received by the standby database by querying the `DBA_LOGSTDBY_LOG` view. For example, to verify that the log file associated with thread number 1 and sequence number 12 was received by the logical standby database, you could repeatedly run the following query on the standby database until it returns the name of the archived redo log file:

```
SQL> SELECT FILE_NAME FROM DBA_LOGSTDBY_LOG WHERE THREAD#=1 AND SEQUENCE#=12
```

8. Verify that all remaining redo log files have been applied by querying the `DBA_LOGSTDBY_PROGRESS` view on the standby database. For example:

```
SQL> SELECT APPLIED_SCN, NEWEST_SCN FROM DBA_LOGSTDBY_PROGRESS;
```

When the numbers in the `APPLIED_SCN` and `NEWEST_SCN` columns are equal, all available data in the redo log files received by the standby database has been applied.

9. Stop SQL Apply on the standby database:

```
SQL> ALTER DATABASE STOP LOGICAL STANDBY APPLY;
```

10. Shut down all active standby database instances with either the `NORMAL` or the `IMMEDIATE` priority. Stop all listeners, agents, and other processes running against the Oracle home:

```
SQL> SHUTDOWN IMMEDIATE;
% agentctl stop
% lsnrctl stop
```

11. Shut down the primary database instance with either the `NORMAL` or the `IMMEDIATE` priority. Stop all listeners, agents, and other processes running against the Oracle home:

```
SQL> SHUTDOWN IMMEDIATE;
% agentctl stop
% lsnrctl stop
```

12. On the primary system, install Oracle Database 10g Release 2 (10.2) into its own Oracle home using the Oracle Universal Installer, as described in the *Oracle Database Upgrade Guide*. To ensure an error-free upgrade, it is recommended that you also install the Companion CD. Do not perform any other upgrade steps.

13. After Oracle Database 10g Release 2 (10.2) has been installed, and with your environment still set to the old (9.2 or 10.1) installation, startup the primary database in `UPGRADE` mode:

```
SQL> STARTUP UPGRADE;
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2=DEFER SCOPE=BOTH;
```

14. From the release 10.2 `ORACLE_HOME`, start the Database Upgrade Assistant and upgrade the primary database. For example:

```
% cd /u01/app/oracle/product/10.2/bin
% ./dbua
```

Note: The old (9.2 or 10.1) database must be included in the `oratab` file to be seen by the Database Upgrade Assistant. For complete information on using DBUA, see the *Oracle Database Upgrade Guide*.

15. Once the database has been upgraded, change your environment to point to the new Oracle Database 10g Release 2 (10.2) installation, shut down the primary database instance, and restart the agent and listener:

```
SQL> SHUTDOWN IMMEDIATE;
% agentctl start
% lsnrctl start
```

16. Startup the primary database instance and enable restricted session to reduce the likelihood of users or applications performing any DML or DDL operations:

```
SQL> STARTUP MOUNT;
SQL> ALTER SYSTEM ENABLE RESTRICTED SESSION;
```

Caution: Do not allow any DML or DDL operations to occur until after restricted session mode is disabled in step 18.

17. Open the primary database and build the LogMiner dictionary:

```
SQL> ALTER DATABASE OPEN;
SQL> EXECUTE DBMS_LOGSTDBY.BUILD;
```

18. Disable restricted session mode on the primary database and archive the current log file:

```
SQL> ALTER SYSTEM DISABLE RESTRICTED SESSION;
SQL> ALTER SYSTEM ARCHIVE LOG CURRENT;
```

19. On the primary database instance, issue the following query to determine the latest LogMiner dictionary build log file:

```
SQL> SELECT NAME FROM V$ARCHIVED_LOG
2> WHERE (SEQUENCE#=(SELECT MAX(SEQUENCE#) FROM V$ARCHIVED_LOG
3> WHERE DICTIONARY_BEGIN = 'YES' AND STANDBY_DEST= 'NO'));
```

Record the name of the log file returned by the query for later reference.

20. On the standby system, install Oracle Database 10g Release 2 (10.2) in its own Oracle home using the Oracle Universal Installer, as described in the *Oracle Database Upgrade Guide*. To ensure an error-free upgrade, it is recommended that you also install the Companion CD. Do not perform any other upgrade steps.
21. After Oracle Database 10g Release 2 (10.2) has been installed, with your environment still set to the old (9.2 or 10.1) installation, startup the logical standby database in UPGRADE mode, activate it, and defer remote archiving:

```
SQL> STARTUP UPGRADE;
SQL> ALTER DATABASE ACTIVATE LOGICAL STANDBY DATABASE;
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2=DEFER SCOPE=BOTH;
```

Caution: Do not allow users to update the activated standby database, because the changes will not be propagated to the primary database.

22. From the 10.2 Oracle_Home directory, start the Database Upgrade Assistant (DBUA) and upgrade the logical standby database.

```
% cd /u01/app/oracle/product/10.2/bin
% ./dbua
```

23. Once the logical standby database has been upgraded, shut down the instance and restart the agent and listener:

```
SQL> SHUTDOWN IMMEDIATE;
% agentctl start
% lsnrctl start
```

24. Copy the latest LogMiner dictionary build log file (identified in step 19) from the primary system to the standby system.

25. Startup the logical standby database instance and turn on the database guard to prevent users from updating objects in the logical standby database:

```
SQL> STARTUP MOUNT;
SQL> ALTER DATABASE GUARD ALL;
SQL> ALTER DATABASE OPEN;
```

26. Register the copied log file on the logical standby database. For example:

```
SQL> ALTER DATABASE REGISTER LOGICAL LOGFILE
2> '/database/LGSTBY/arch/arch1_48.arc';
```

27. Start SQL Apply on the logical standby database:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY INITIAL;
```

Note: The above command will initially fail with the following error:

```
ORA-16101: a valid start SCN could not be found
```

To resolve the error, register the logical log file as described in step 26 and reissue the statement to start SQL Apply.

28. If you are using Real Application Clusters, startup the other standby database instances.

29. On the primary system, enable archiving to the upgraded logical standby database.

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2=ENABLE;
```

30. If you are using Real Application Clusters, startup the other primary database instances.

Data Type and DDL Support on a Logical Standby Database

When setting up a logical standby database, you must ensure the logical standby database can maintain the data types and tables in your primary database. This appendix lists the various database objects, storage types, and PL/SQL supplied packages that are supported and unsupported by logical standby databases. It contains the following topics:

- [Data Type Considerations](#)
- [Storage Type Considerations](#)
- [PL/SQL Supplied Packages Considerations](#)
- [Unsupported Tables, Sequences, and Views](#)
- [Skipped SQL Statements on a Logical Standby Database](#)
- [DDL Statements Supported by a Logical Standby Database](#)

C.1 Data Type Considerations

The following sections list the supported and unsupported database objects:

- [Supported Data Types in a Logical Standby Database](#)
- [Unsupported Data Types in a Logical Standby Database](#)

C.1.1 Supported Data Types in a Logical Standby Database

Logical standby databases support the following data types:

BINARY_DOUBLE
BINARY_FLOAT
BLOB
CHAR
CLOB and NCLOB
DATE
INTERVAL YEAR TO MONTH
INTERVAL DAY TO SECOND
LONG
LONG RAW
NCHAR
NUMBER
NVARCHAR2
RAW

TIMESTAMP
TIMESTAMP WITH LOCAL TIMEZONE
TIMESTAMP WITH TIMEZONE
VARCHAR2 and VARCHAR

Note: SQL Apply support for the following data types has compatibility requirements on the primary database:

- Multibyte CLOB support (requires primary database to run at a compatibility of 10.1 or higher).
 - IOT support without LOBs and Overflows (requires primary database to run at a compatibility of 10.1 or higher);
 - IOT support with LOB and Overflow (requires primary database to run at a compatibility of 10.2 or higher)
-
-

C.1.2 Unsupported Data Types in a Logical Standby Database

Logical standby databases do not support the following data types:

BFILE
Collections (including VARRAYS and nested tables)
Encrypted columns
Multimedia data types (including Spatial, Image, and Context)
ROWID, UROWID
User-defined types
XMLType

C.2 Storage Type Considerations

The following sections list the supported and unsupported storage types:

- [Support Storage Types](#)
- [Unsupported Storage Type](#)

C.2.1 Support Storage Types

Logical standby databases support the following storage types:

Cluster tables (including index clusters and heap clusters)
Index-organized tables (partitioned and nonpartitioned, including overflow segments)
Heap-organized tables (partitioned and nonpartitioned)

C.2.2 Unsupported Storage Type

Logical standby databases do not support the segment compression storage type.

C.3 PL/SQL Supplied Packages Considerations

The following sections list the supported and unsupported PL/SQL supplied packages:

- [Supported PL/SQL Supplied Packages](#)
- [Unsupported PL/SQL Supplied Packages](#)

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

C.3.1 Supported PL/SQL Supplied Packages

Oracle PL/SQL supplied packages that do not modify system metadata or user data leave no footprint in the archived redo log files, and hence are safe to use on the primary database. Examples of such packages are DBMS_OUTPUT, DBMS_RANDOM, DBMS_PIPE, DBMS_DESCRIBE, DBMS_OBFUSCATION_TOOLKIT, DBMS_TRACE, and DBMS_METADATA.

Oracle PL/SQL supplied packages that do not modify system metadata but may modify user data are supported by SQL Apply, as long as the modified data belongs to the supported data types listed in [Section C.1.1](#). Examples of such packages are DBMS_LOB, DBMS_SQL, and DBMS_TRANSACTION.

C.3.2 Unsupported PL/SQL Supplied Packages

Oracle PL/SQL supplied packages that modify system metadata typically are not supported by SQL Apply, and therefore their effects are not visible on the logical standby database. Examples of such packages are DBMS_JAVA, DBMS_REGISTRY, DBMS_ALERT, DBMS_SPACE_ADMIN, DBMS_REFRESH, DBMS_REDEFINITION, DBMS_SCHEDULER, and DBMS_AQ.

Specific support for DBMS_JOB has been provided. Job execution is suspended on a logical standby database and jobs cannot be scheduled directly on the standby database. However, jobs submitted on the primary database are replicated in the standby database. In the event of a switchover or failover, jobs scheduled on the original primary database will automatically begin running on the new primary database.

C.4 Unsupported Tables, Sequences, and Views

It is important to identify unsupported database objects on the primary database before you create a logical standby database. This is because changes made to unsupported data types, tables, sequences, or views on the primary database will be automatically skipped by SQL Apply on the logical standby database. Moreover, no error message will be returned.

Some schemas that ship with the Oracle database are skipped by SQL Apply. To determine exactly which schemas will be skipped, query the DBA_LOGSTDBY_SKIP view.

```
SELECT OWNER FROM DBA_LOGSTDBY_SKIP WHERE STATEMENT_OPT = 'INTERNAL SCHEMA';
```

To determine if the primary database contains unsupported objects, query the DBA_LOGSTDBY_UNSUPPORTED view. See [Chapter 16, "Views Relevant to Oracle Data Guard"](#) for more information about views.

For example, use the following query on the primary database to list the schema and table names of primary database tables that are not supported by logical standby databases:

```
SQL> SELECT DISTINCT OWNER, TABLE_NAME FROM DBA_LOGSTDBY_UNSUPPORTED
      2> ORDER BY OWNER, TABLE_NAME;
```

OWNER	TABLE_NAME
HR	COUNTRIES

```
OE          ORDERS
OE          CUSTOMERS
OE          WAREHOUSES
```

To view the column names and data types for one of the tables listed in the previous query, use a SELECT statement similar to the following:

```
SQL> SELECT COLUMN_NAME, DATA_TYPE FROM DBA_LOGSTDBY_UNSUPPORTED
      2> WHERE OWNER='OE' AND TABLE_NAME = 'CUSTOMERS';
```

COLUMN_NAME	DATA_TYPE
CUST_ADDRESS	CUST_ADDRESS_TYP
PHONE_NUMBERS	PHONE_LIST_TYP
CUST_GEO_LOCATION	SDO_GEOMETRY

If the primary database contains unsupported tables, SQL Apply automatically excludes these tables when applying redo data to the logical standby database.

Note: If you determine that the critical tables in your primary database will not be supported on a logical standby database, then you might want to consider using a physical standby database. Physical standby databases do not have any such data type restrictions.

C.5 Skipped SQL Statements on a Logical Standby Database

By default, the following SQL statements are automatically skipped by SQL Apply:

```
ALTER DATABASE
ALTER MATERIALIZED VIEW
ALTER MATERIALIZED VIEW LOG
ALTER SESSION
ALTER SYSTEM
CREATE CONTROL FILE
CREATE DATABASE
CREATE DATABASE LINK
CREATE PFILE FROM SPFILE
CREATE MATERIALIZED VIEW
CREATE MATERIALIZED VIEW LOG
CREATE SCHEMA AUTHORIZATION
CREATE SPFILE FROM PFILE
DROP DATABASE LINK
DROP MATERIALIZED VIEW
DROP MATERIALIZED VIEW LOG
EXPLAIN
LOCK TABLE
SET CONSTRAINTS
SET ROLE
SET TRANSACTION
```

All other SQL statements executed on the primary database are applied to the logical standby database.

C.6 DDL Statements Supported by a Logical Standby Database

The following tables list the supported values for the `stmt` parameter of the `DBMS_LOGSTDBY.SKIP` procedure and the statement options for skipping SQL DDL statements:

- [Table C–1, "Values for stmt Parameter of the DBMS_LOGSTDBY.SKIP procedure"](#)
- [Table C–2, "Statement Options for Skipping SQL DDL Statements"](#)

See Also: *Oracle Database PL/SQL Packages and Types Reference* for complete information about the `DBMS_LOGSTDBY` package and [Section 9.4.4, "Setting up a Skip Handler for a DDL Statement"](#)

[Table C–1](#) lists the supported values for the `stmt` parameter of the `DBMS_LOGSTDBY.SKIP` procedure. The left column of the table lists the keywords that may be used to identify the set of SQL statements to the right of the keyword. Any of the SQL statements in the right column, however, are also valid values. Note that keywords are generally defined by database object.

Table C–1 Values for `stmt` Parameter of the `DBMS_LOGSTDBY.SKIP` procedure

Keyword	Associated SQL Statements
DML	Includes DML statements on a table (for example: INSERT, UPDATE, and DELETE)
CLUSTER	CREATE CLUSTER AUDIT CLUSTER DROP CLUSTER TRUNCATE CLUSTER
CONTEXT	CREATE CONTEXT DROP CONTEXT
DATABASE LINK	CREATE DATABASE LINK DROP DATABASE LINK
DIMENSION	CREATE DIMENSION ALTER DIMENSION DROP DIMENSION
DIRECTORY	CREATE DIRECTORY DROP DIRECTORY
INDEX	CREATE INDEX ALTER INDEX DROP INDEX
NON_SCHEMA_DDL	<i>All DDL that does not pertain to a particular schema</i>
PROCEDURE ¹	CREATE FUNCTION CREATE LIBRARY CREATE PACKAGE CREATE PACKAGE BODY CREATE PROCEDURE DROP FUNCTION DROP LIBRARY DROP PACKAGE DROP PROCEDURE
PROFILE	CREATE PROFILE ALTER PROFILE DROP PROFILE

Table C-1 (Cont.) Values for stmt Parameter of the DBMS_LOGSTDBY.SKIP procedure

Keyword	Associated SQL Statements
PUBLIC DATABASE LINK	CREATE PUBLIC DATABASE LINK DROP PUBLIC DATABASE LINK
PUBLIC SYNONYM	CREATE PUBLIC SYNONYM DROP PUBLIC SYNONYM
ROLE	CREATE ROLE ALTER ROLE DROP ROLE SET ROLE
ROLLBACK STATEMENT	CREATE ROLLBACK SEGMENT ALTER ROLLBACK SEGMENT DROP ROLLBACK SEGMENT
SCHEMA_DDL	<i>All DDL statements that create, modify, or drop schema objects (for example: tables, indexes, and columns)</i>
SEQUENCE	CREATE SEQUENCE DROP SEQUENCE
SESSION	Log-ons
SYNONYM	CREATE SYNONYM DROP SYNONYM
SYSTEM AUDIT	AUDIT <i>SQL_statements</i> NOAUDIT <i>SQL_statements</i>
SYSTEM GRANT	GRANT <i>system_privileges_and_roles</i> REVOKE <i>system_privileges_and_roles</i>
TABLE	CREATE TABLE DROP TABLE TRUNCATE TABLE
TABLESPACE	CREATE TABLESPACE DROP TABLESPACE TRUNCATE TABLESPACE
TRIGGER	CREATE TRIGGER ALTER TRIGGER with ENABLE and DISABLE clauses DROP TRIGGER ALTER TABLE with ENABLE ALL TRIGGERS clause ALTER TABLE with DISABLE ALL TRIGGERS clause
TYPE	CREATE TYPE CREATE TYPE BODY ALTER TYPE DROP TYPE DROP TYPE BODY
USER	CREATE USER ALTER USER DROP USER
VIEW	CREATE VIEW DROP VIEW

¹ Java schema objects (sources, classes, and resources) are considered the same as procedure for purposes of skipping (ignoring) SQL statements.

Table C-2 lists the statement options for skipping SQL DDL statements.

Table C-2 Statement Options for Skipping SQL DDL Statements

Statement Option	SQL Statements and Operations
ALTER SEQUENCE	ALTER SEQUENCE
ALTER TABLE	ALTER TABLE
COMMENT TABLE	COMMENT ON TABLE <i>table</i> , <i>view</i> , <i>materialized view</i> COMMENT ON COLUMN <i>table.column</i> , <i>view.column</i> , <i>materialized_view.column</i>
DELETE TABLE	DELETE FROM <i>table</i> , <i>view</i>
EXECUTE PROCEDURE	CALL Execution of any procedure or function or access to any variable, library, or cursor inside a package.
GRANT DIRECTORY	GRANT <i>privilege</i> ON <i>directory</i> REVOKE <i>privilege</i> ON <i>directory</i>
GRANT PROCEDURE	GRANT <i>privilege</i> ON <i>procedure</i> , <i>function</i> , <i>package</i> REVOKE <i>privilege</i> ON <i>procedure</i> , <i>function</i> , <i>package</i>
GRANT SEQUENCE	GRANT <i>privilege</i> ON <i>sequence</i> REVOKE <i>privilege</i> ON <i>sequence</i>
GRANT TABLE	GRANT <i>privilege</i> ON <i>table</i> , <i>view</i> , <i>materialized view</i> REVOKE <i>privilege</i> ON <i>table</i> , <i>view</i> , <i>materialized view</i>
GRANT TYPE	GRANT <i>privilege</i> ON TYPE REVOKE <i>privilege</i> ON TYPE
INSERT TABLE	INSERT INTO <i>table</i> , <i>view</i>
LOCK TABLE	LOCK TABLE <i>table</i> , <i>view</i>
SELECT SEQUENCE	Any statement containing <i>sequence.CURRVAL</i> or
SELECT TABLE	SELECT FROM <i>table</i> , <i>view</i> , <i>materialized view</i> REVOKE <i>privilege</i> ON <i>table</i> , <i>view</i> , <i>materialized view</i>
UPDATE TABLE	UPDATE <i>table</i> , <i>view</i>

See Also: The following sections that provide usage examples of the SKIP and UNSKIP options:

- [Section 9.4.3, "Using DBMS_LOGSTDBY.SKIP to Prevent Changes to Specific Schema Objects"](#)
- [Section 9.4.4, "Setting up a Skip Handler for a DDL Statement"](#)
- [Section 9.4.5, "Modifying a Logical Standby Database"](#)
- [Section 9.4.6, "Adding or Re-Creating Tables On a Logical Standby Database"](#)

Data Guard and Real Application Clusters

An Oracle Data Guard configuration can consist of any combination of single-instance and RAC multiple-instance databases. This chapter summarizes the configuration requirements and considerations that apply when using Oracle Data Guard with Oracle Real Application Clusters databases. It contains the following sections:

- [Configuring Standby Databases in a Real Application Clusters Environment](#)
- [Configuration Considerations in a Real Application Clusters Environment](#)
- [Troubleshooting](#)

D.1 Configuring Standby Databases in a Real Application Clusters Environment

You can configure a standby database to protect a primary database using Real Application Clusters. The following table describes the possible combinations of instances in the primary and standby databases:

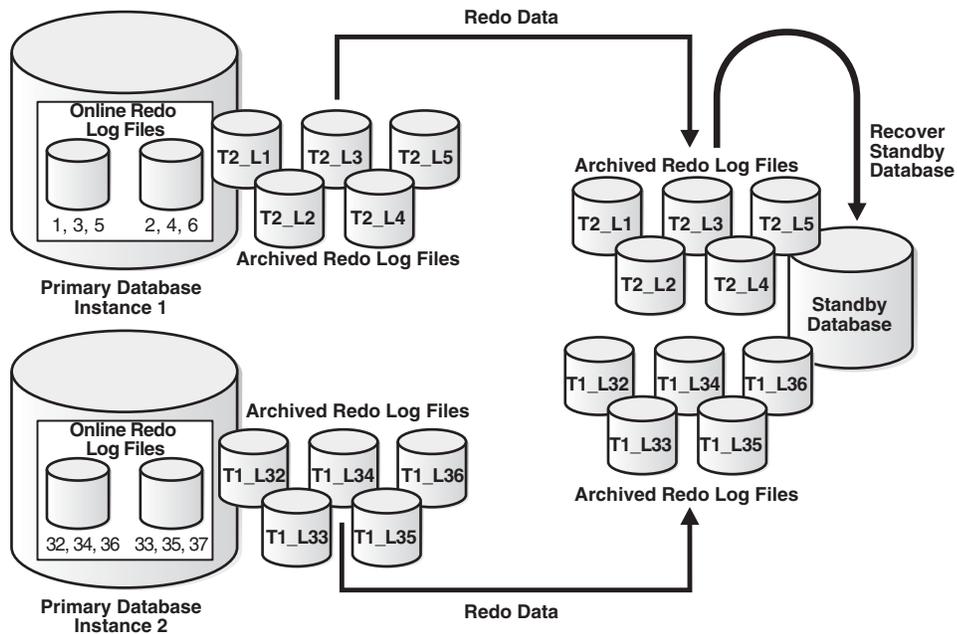
Instance Combinations	Single-Instance Standby Database	Multi-Instance Standby Database
Single-instance primary database	Yes	Yes
Multi-instance primary database	Yes	Yes

In each scenario, each instance of the primary database transmits its redo data to an instance of the standby database.

D.1.1 Setting Up a Multi-Instance Primary with a Single-Instance Standby

[Figure D-1](#) illustrates a Real Application Clusters database with two primary database instances (a multi-instance primary database) transmitting redo data to a single-instance standby database.

Figure D-1 Transmitting Redo Data from a Multi-Instance Primary Database



In this case, Instance 1 of the primary database archives redo data to local archived redo log files 1, 2, 3, 4, 5 and transmits the redo data to the standby database destination, while Instance 2 archives redo data to local archived redo log files 32, 33, 34, 35, 36 and transmits the redo data to the same standby database destination. The standby database automatically determines the correct order in which to apply the archived redo log files.

To set up a primary database in a Real Application Clusters environment

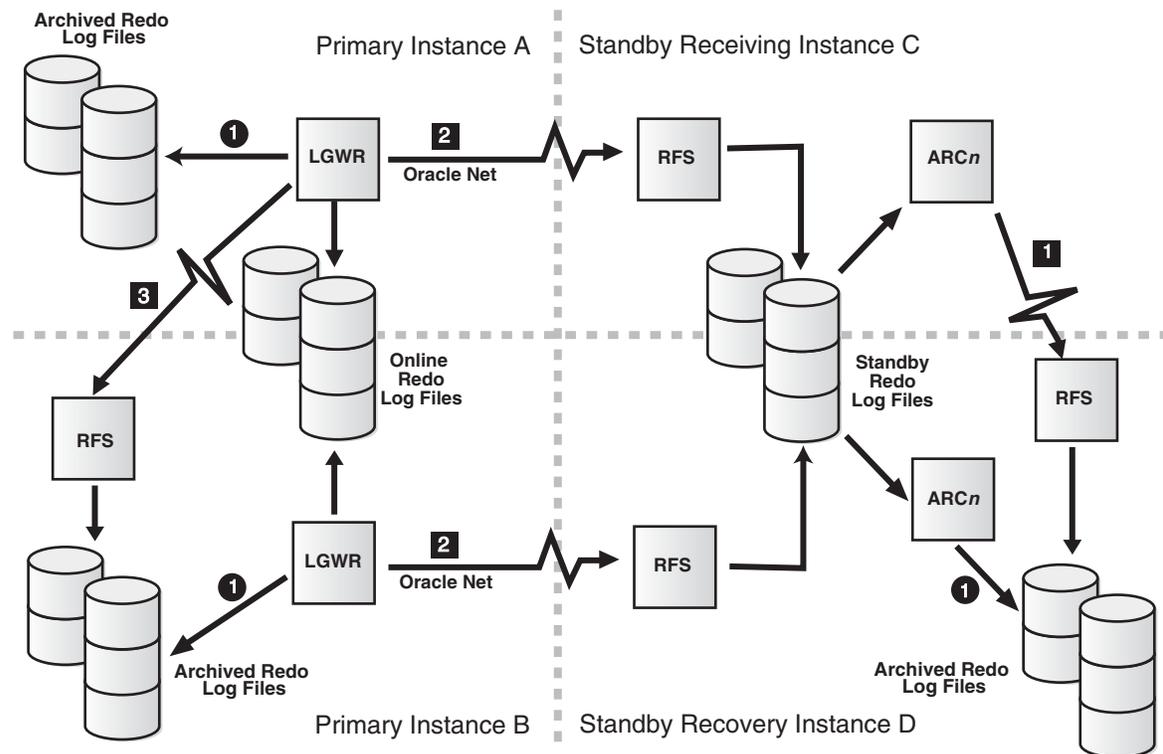
Follow the instructions in [Chapter 3](#) (for physical standby database creation) or [Chapter 4](#) (for logical standby database creation) to configure each primary instance.

To set up a single instance standby database

Follow the instructions in [Chapter 3](#) (for physical standby database creation) or [Chapter 4](#) (for logical standby database creation) to define the `STANDBY_ARCHIVE_DEST` and `LOG_ARCHIVE_FORMAT` parameters to specify the location of the archived redo log files and standby redo log files.

D.1.2 Setting Up a Multi-Instance Primary with a Multi-Instance Standby

[Figure D-2](#) shows a configuration where the primary and standby databases are in a Real Application Clusters environment. This enables you to separate the redo transport services processing from the log apply services processing on the standby database, thereby improving overall primary and standby database performance.

Figure D–2 Standby Database in Real Application Clusters

In [Figure D–2](#), the numbers within circles indicate local connections, and the numbers within boxes indicate remote connections.

In a Real Application Clusters environment, any standby instance can receive redo data from the primary database; this is a **receiving instance**. However, the archived redo log files must ultimately reside on disk devices accessible by the **recovery instance**. Transferring the standby database archived redo log files from the receiving instance to the recovery instance is achieved using the cross-instance archival operation.

The standby database cross-instance archival operation requires use of standby redo log files as the temporary repository of primary database archived redo log files. Using standby redo log files not only improves standby database performance and reliability, but also allows the cross-instance archival operation to be performed on clusters that do not have a cluster file system. However, because standby redo log files are required for the cross-instance archival operation, the primary database can use either the log writer process (LGWR) or archiver processes (ARCn) to perform the archival operations on the primary database.

When both the primary and standby databases are in a Real Application Clusters configuration, then a single instance of the standby database applies all sets of log files transmitted by the primary instances. In this case, the standby instances that are *not* applying redo data cannot be in read-only mode while Redo Apply is in progress.

To set up a standby database in a Real Application Clusters environment

Perform the following steps to set up redo transport services on the standby database:

1. Create the standby redo log files. In a Real Application Clusters environment, the standby redo log files must reside on disk devices shared by all instances. See [Section 3.1.3](#) for more information.

2. On the recovery instance, define the `LOCATION` attribute of the `LOG_ARCHIVE_DEST_1` initialization parameter to archive locally, because cross-instance archiving is not necessary.
3. On the receiving instance, define the `SERVICE` attribute of the `LOG_ARCHIVE_DEST_1` initialization parameter to archive to the recovery instance.
4. Start log apply services on the recovery instance.

To set up a primary database in a Real Application Clusters environment

Perform the following steps to set up redo transport services on the primary database:

1. On all instances, define the `LGWR` attribute on the `LOG_ARCHIVE_DEST_n` parameter to designate that the `LGWR` process will perform the archival operation.
2. Configure each standby instance to send redo data to the receiving instance by setting the `LOG_ARCHIVE_DEST_n` parameter to an appropriate value.

Ideally, each primary database instance should archive to a corresponding standby database instance. However, this is not required.

D.2 Configuration Considerations in a Real Application Clusters Environment

This section contains the Data Guard configuration information that is specific to Real Application Clusters environments. It contains the following topics:

- [Format for Archived Redo Log Filenames](#)
- [Archive Destination Quotas](#)
- [Data Protection Modes](#)
- [Role Transitions](#)

D.2.1 Format for Archived Redo Log Filenames

The format for archived redo log filenames is in the form of `log_%%parameter`, where `%%parameter` can include one or more of the parameters in [Table D-1](#).

Table D-1 Directives for the LOG_ARCHIVE_FORMAT Initialization Parameter

Directives	Description
%a	Database activation ID.
%A	Database activation ID, zero filled.
%d	Database ID.
%D	Database ID, zero filled.
%t	Instance thread number.
%T	Instance thread number, zero filled.
%s	Log file sequence number.
%S	Log file sequence number, zero filled.
%r	Resetlogs ID.
%R	Resetlogs ID, zero filled.

For example:

```
LOG_ARCHIVE_FORMAT = log%d_%t_%s_%r.arc
```

The thread parameters %t or %T are mandatory for Real Application Clusters to uniquely identify the archived redo log files with the LOG_ARCHIVE_FORMAT parameter. See [Section 5.7.1](#) for more information about storage locations for archived redo log files.

D.2.2 Archive Destination Quotas

You can specify the amount of physical storage on a disk device to be available for an archiving destination using the QUOTA_SIZE attribute of the LOG_ARCHIVE_DEST_n initialization parameter. An archive destination can be designated as being able to occupy all or some portion of the physical disk represented by the destination. For example, in a Real Application Clusters environment, a physical disk device can be shared by two or more separate nodes. As there is no cross-instance initialization parameter knowledge, none of the Real Application Clusters nodes is aware that the physical disk device is shared with other instances. This leads to substantial problems when the destination disk device becomes full; the error is not detected until every instance tries to archive to the already full device. This affects database availability.

D.2.3 Data Protection Modes

In a Real Application Clusters configuration when running in either maximum protection or maximum availability mode, any instance that loses connectivity with a standby destination will cause all other instances to stop sending data to that destination (this maintains the integrity of the data that has been transmitted to that destination).

When the failed standby destination comes back up, Data Guard runs the site in resynchronization mode until no gaps remain. Then, the standby destination can participate in the Data Guard configuration again.

The following list describes the behavior of the protection modes in Real Application Clusters environments:

- Maximum protection configuration

If a lost destination is the *last* participating LGWR SYNC destination, the instance loses connectivity and will be shut down. Other instances in a Real Application Clusters configuration that still have connectivity to the standby destinations will recover the lost instance and continue sending to their standby destinations. Only when every instance in a Real Application Clusters configuration loses connectivity to the last standby destination will the primary database be shut down.

D.2.4 Role Transitions

This section contains the following topics:

- [Switchovers](#)
- [Failovers](#)

D.2.4.1 Switchovers

For a Real Application Clusters database, only one primary instance and one standby instance can be active during a switchover. Therefore, before a switchover, shut down all but one primary instance and one standby instance. After the switchover

completes, restart the primary and standby instances that were shut down during the switchover.

Note: The SQL `ALTER DATABASE` statement used to perform the switchover automatically creates redo log files if they do not already exist. Because this can significantly increase the time required to complete the `COMMIT` operation, Oracle recommends that you manually add redo log files when creating physical standby databases.

D.2.4.2 Failovers

Before performing a failover to a Real Application Clusters standby database, first shut down all but one standby instance. After the failover completes, restart the instances that were shut down.

D.3 Troubleshooting

This section provides help troubleshooting problems with Real Application Clusters. It contains the following sections:

- [Switchover Fails in a Real Application Clusters Configuration](#)
- [Avoiding Downtime in Real Application Clusters During a Network Outage](#)

D.3.1 Switchover Fails in a Real Application Clusters Configuration

When your database is using Real Application Clusters, active instances prevent a switchover from being performed. When other instances are active, an attempt to switch over fails with the following error message:

```
SQL> ALTER DATABASE COMMIT TO SWITCHOVER TO STANDBY;
ALTER DATABASE COMMIT TO SWITCHOVER TO STANDBY *
ORA-01105: mount is incompatible with mounts by other instances
```

Action: Query the `GV$INSTANCE` view as follows to determine which instances are causing the problem:

```
SQL> SELECT INSTANCE_NAME, HOST_NAME FROM GV$INSTANCE
       2> WHERE INST_ID <> (SELECT INSTANCE_NUMBER FROM V$INSTANCE);
INSTANCE_NAME HOST_NAME
-----
INST2          standby2
```

In the previous example, the identified instance must be manually shut down before the switchover can proceed. You can connect to the identified instance from your instance and issue the `SHUTDOWN` statement remotely, for example:

```
SQL> CONNECT SYS/CHANGE_ON_INSTALL@standby2 AS SYSDBA
SQL> SHUTDOWN;
SQL> EXIT
```

D.3.2 Avoiding Downtime in Real Application Clusters During a Network Outage

If you configured Data Guard to support a primary database in a Real Application Clusters environment and the primary database is running in maximum protection mode, a network outage between the primary database and all of its standby databases will disable the primary database until the network connection is restored.

The maximum protection mode dictates that if the last standby database becomes unavailable, processing halts on the primary database.

If you expect the network to be down for an extended period of time, consider changing the primary database to run in either the maximum availability or the maximum performance mode until network connectivity is restored. If you change the primary database to maximum availability mode, it is possible for there to be a lag between the primary and standby databases, but you gain the ability to use the primary database until the network problem is resolved.

If you choose to change the primary database to the maximum availability mode, it is important to use the following procedures to prevent damage to your data.

The following steps describe what to do if the network goes down and you want to change the protection mode for the Real Application Clusters configuration. The example assumes you are using a server parameter file (SPFILE), not a PFILE.

1. At this point all Real Application Clusters primary instances are shut down. Issue the `STARTUP MOUNT` command to start one instance:

```
STARTUP MOUNT;
```

2. Follow the instructions in [Section 5.6.2](#) (or, if you are using the broker, see *Oracle Data Guard Broker*) to change the mode from the maximum protection mode to either maximum availability or maximum performance mode. For example, the following statement sets the maximum availability protection mode:

```
ALTER DATABASE SET STANDBY DATABASE TO MAXIMIZE AVAILABILITY;
```

3. Open the Real Application Clusters primary database for general access.

Later, when the network comes back up, perform the following steps to revert to the maximum protection mode:

1. Shut down all instances of the Real Application Clusters primary database.
2. Mount a single instance of the Real Application Clusters primary database, without opening it for general access.
3. Change mode on the Real Application Clusters primary database from its current (maximum availability or maximum performance) mode to the maximum protection mode.
4. Open the Real Application Clusters primary database for general access.

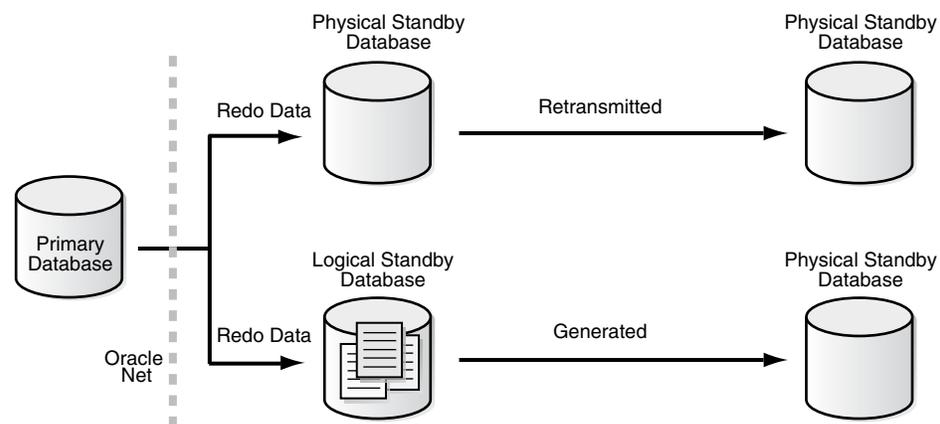
Cascaded Destinations

To reduce the load on your primary system, you can implement **cascaded destinations**, whereby a standby database receives its redo data from another standby database, instead of directly from the primary database. You can configure:

- A physical standby database to retransmit the incoming redo data it receives from the primary database to other remote destinations in the same manner as the primary database
- A logical standby database (because it is open in read/write mode) to send the redo data it generates (after filtering and applying the redo data it receives from the primary database) to its own set of standby (physical or logical) databases

Figure E-1 shows cascaded destinations to physical and logical standby databases.

Figure E-1 Cascaded Destination Configuration Example



A standby database can cascade redo data to up to nine destinations. However, from a practical perspective, only standby databases primarily intended to off-load reporting or backups typically would be configured to receive cascaded redo data. Standby databases that could potentially be involved in role transitions typically are configured to receive redo data directly from the primary database and have `VALID_FOR` attribute on the `LOG_ARCHIVE_DEST_n` parameter defined so that when a switchover or failover operation occurs, redo data continues to be received directly from the *new* primary database.

This appendix contains the following sections:

- [Configuring Cascaded Destinations](#)
- [Role Transitions with Cascaded Destinations](#)

- [Examples of Cascaded Destinations](#)

E.1 Configuring Cascaded Destinations

The following sections describe how to set up a Data Guard configuration to use cascaded destinations:

- [Configuring Cascaded Destinations for Physical Standby Databases](#)
- [Configuring Cascaded Destinations for Logical Standby Databases](#)

E.1.1 Configuring Cascaded Destinations for Physical Standby Databases

To enable a physical standby database to send the incoming redo data to another set of destinations, you must define the following items:

- Define the `LOG_ARCHIVE_DEST_n` initialization parameter on the primary database to set up a physical standby database that will be the starting point for a cascade. Define the destination to use:
 - The `ARCH` or the `LGWR` attributes to specify the transport method. With the `LGWR` transport method, you can use either `SYNC` or `ASYNC` network protocols, depending on your requirements.
 - The `VALID_FOR` attribute to handle the cascaded destinations and the original primary and standby destinations. Define destinations for the primary database and other standby databases as well as the cascading standby databases.
- On the receiving physical standby database, define more than the usual number of standby redo log files and ensure archiving is enabled.

At this point, you can begin defining the `LOG_ARCHIVE_DEST_n` initialization parameter on the physical standby database that will define the end points of the cascade.

[Table E-1](#) shows the initialization parameters for the Boston primary database, which sends redo to the Chicago physical standby database, which cascades its archived redo log files to the Denver standby database. In the example, the Denver database is a logical standby database, but note that a physical standby database can cascade redo to either a physical or a logical standby database.

Table E–1 Initialization Parameters for Primary, Physical, and Logical Standby Databases

Boston Database (Primary Role)	Chicago Database (Standby Role)	Denver Database (Standby Role)
DB_UNIQUE_NAME=boston	DB_UNIQUE_NAME=chicago	DB_UNIQUE_NAME=denver
LOG_ARCHIVE_CONFIG= 'DG_CONFIG=(chicago,boston,denver)'	LOG_ARCHIVE_CONFIG= 'DG_CONFIG=(chicago,boston,denver)'	LOG_ARCHIVE_CONFIG= 'DG_CONFIG=(chicago,boston,denver)'
LOG_ARCHIVE_DEST_1= 'LOCATION=/arch1/boston/ VALID_FOR=(ONLINE_ LOGFILES,PRIMARY_ROLE) DB_UNIQUE_NAME=boston'	LOG_ARCHIVE_DEST_1= 'LOCATION=/arch1/chicago/ VALID_FOR=(ONLINE_ LOGFILES,ALL_ROLES) DB_UNIQUE_NAME=chicago'	LOG_ARCHIVE_DEST_1= 'LOCATION=/arch1/denver/ VALID_FOR=(ONLINE_ LOGFILES,ALL_ROLES) DB_UNIQUE_NAME=denver'
LOG_ARCHIVE_DEST_2= 'SERVICE=denver VALID_FOR=(STANDBY_ LOGFILES,STANDBY_ROLE) DB_UNIQUE_NAME=denver'	LOG_ARCHIVE_DEST_2= 'SERVICE=denver VALID_FOR=(STANDBY_ LOGFILES,STANDBY_ROLE) DB_UNIQUE_NAME=denver'	LOG_ARCHIVE_DEST_2= 'LOCATION=/arch2/denver/ VALID_FOR=(STANDBY_ LOGFILES,STANDBY_ROLE) DB_UNIQUE_NAME=denver'
LOG_ARCHIVE_DEST_3= 'SERVICE=chicago VALID_FOR= (ONLINE_LOGFILES,PRIMARY_ROLE) DB_UNIQUE_NAME=chicago'	LOG_ARCHIVE_DEST_3= 'SERVICE=boston VALID_FOR= (ONLINE_LOGFILES,PRIMARY_ROLE) DB_UNIQUE_NAME=boston'	STANDBY_ARCHIVE_DEST=/arch2/denver/ REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE
STANDBY_ARCHIVE_DEST=/arch1/boston/ REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE	STANDBY_ARCHIVE_DEST=/arch1/chicago/ REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE	

Both the Boston primary database and the Chicago physical standby database define the LOG_ARCHIVE_DEST_2 initialization parameter as 'SERVICE=denver VALID_FOR=(STANDBY_LOGFILES, STANDBY_ROLE). Hence, even if the Boston and Chicago databases switch roles, the redo data will continue to be cascaded to the Denver database. Remember, as part of the original setup of the physical standby database, you should define a local destination, VALID_FOR=(ONLINE_LOGFILE, PRIMARY_ROLE), that will be used for local archiving when the physical standby database transitions to the primary role.

E.1.2 Configuring Cascaded Destinations for Logical Standby Databases

A logical standby database that receives redo data directly from the primary database can be configured to cascade the redo data it generates (after it has filtered and applied the redo data it receives from the primary database) to other standby databases. Because redo data cascaded from a logical standby database is not identical to the redo data originally generated by the primary database, it cannot be applied to any standby database created directly from the primary database. Instead, any standby databases that receive cascaded redo data from a logical standby database must be created from a copy of the logical standby database, and the following will be true:

- Physical standby databases created from a logical standby database will be a block-for-block copy of the logical standby database and a logical copy of the original primary database.
- Logical standby databases created from a logical standby database will be logical copies of the parent logical standby database and might bear only a partial resemblance to the original primary database. This is because the original primary database's data is there and so is anything else stored in the parent logical standby database including any other changes such as different indexes or materialized views.

For standby databases that receive cascaded redo data from a logical standby database, you must perform the same setup tasks as for a physical or logical standby database that receives redo data directly from the primary database.

As with physical standby databases, you can use any transport mode (LGWR or ARCH) and network protocol (SYNC or ASYNC). You must configure standby redo log files on your standby databases.

E.2 Role Transitions with Cascaded Destinations

Most role transitions can be performed involving standby databases that receive redo log files cascaded from another standby database. However, to minimize risk of data loss and ensure the fastest possible role transition, Oracle recommends that any standby databases that are primarily configured for disaster-recovery purposes receive redo data directly from the primary database.

E.2.1 Standby Databases Receiving Redo Data from a Physical Standby Database

The process to perform a switchover or failover is exactly the same in a cascaded configuration, because all physical standby databases that receive retransmitted primary database redo data are identical and valid for role transitions. The only difference is additional time may be required for the end-of-redo data to cascade to the standby database. See [Section 7.2](#) for information about performing role transitions with a physical standby database.

E.2.2 Standby Databases Receiving Redo Data from a Logical Standby Database

Any standby database that receives redo data cascaded from a logical standby database cannot participate in a switchover involving the primary database. (Only logical standby databases that receive redo data directly from the primary database can participate in switchovers.) If you fail over to a database that receives redo data generated by a logical standby database, then only other logical standby databases that receive redo data cascaded from the same logical standby database are able to continue to participate in the Data Guard configuration after the failover. See [Section 7.3](#) for information about performing role transitions with a logical standby database.

E.3 Examples of Cascaded Destinations

The following scenarios demonstrate configuration options and uses for cascaded destinations.

E.3.1 Local Physical Standby and Cascaded Remote Physical Standby

You have a primary database in your corporate offices, and you want to create a standby database in another building on your local area network (LAN). In addition, you have a legal insurance requirement to keep the redo data and backup copies off site at a geographically distant location outside of your LAN but on your wide area network (WAN).

You could define two destinations on your primary database so that redo data could be transmitted to both of these sites, but this would put an extra workload on your primary database throughput due to the network latency of sending the redo data over the WAN.

To solve this problem, you could define a tight connection between your primary and physical standby databases in your LAN using the LGWR and SYNC network transports and standby redo log files. This would protect against losing access to the primary database, and it provides an alternate site for production when maintenance is required on the primary database. The secondary location on the WAN could be

serviced by the physical standby database, ensuring that the redo data is stored off site. Nightly backups on the production database could then be moved to the WAN remote standby database, which removes the requirement to ship tapes to the off-site storage area.

Finally, in a worst case scenario, where you lose access to both the primary database and the physical standby database on the LAN, you could fail over to the remote standby database with minimal data loss. If you can gain access to the redo log file of the last standby database from the original standby database, you could recover it on the remote standby database, incurring no data loss.

The only time you would incur problems by sending the information over the WAN is during a switchover or failover, when the physical standby database has transitioned to the primary role. However, this configuration would still meet your insurance requirements.

E.3.2 Local Physical Standby and Cascaded Remote Logical Standby

You have a primary database in a remote city, and you would like to have access to its data locally for reporting purposes. The primary database already has a standby database set up for failure protection in an off-site location on the LAN. Putting a destination on the primary database to send the information to your site would adversely affect the performance of the primary database.

Solving this problem is similar to the solution that is described in scenario 1, except that you are sending the redo data to a logical standby database, because you already have a physical standby database. First, ensure standby log files are being used.

If standby redo log files are not defined, you can define them dynamically on the standby database. The standby database will begin using the standby redo log files after the next log switch on the primary database.

Next, perform the normal setup tasks for a logical standby database. Any of the steps required to prepare to use a logical standby database must be done at the primary location as usual. After the logical standby database is up and running, define your destination parameters on the physical standby database to send the redo data over the WAN, where it will be applied to the logical standby database.

E.3.3 Local and Remote Physical Standby and Cascaded Local Logical Standby

A primary database located in a manufacturing site already is configured with two physical standby databases. One standby database is located on the LAN in another building, and the second standby database is more remotely located on a WAN in the corporate offices. You cannot use cascaded destinations for the standby database on the WAN, because there is a requirement to have two standby databases in no-data-loss mode. Also, the marketing department requested access to the manufacturing data for sales predictions. The marketing department needs access to the data on a daily basis, and they want to combine sales data with manufacturing data to better understand sales versus the actual manufacturing times.

One solution would be to allow marketing to access a physical standby database in the corporate offices using read-only mode. However, putting the standby database in read-only mode requires stopping Redo Apply. This means that the physical standby database can only catch up with the primary database at night, while it is still receiving data from the second and third shifts at the manufacturing plant. In addition, the standby database would always be at least 12 hours behind in applying archived redo log files. You could add another destination to the primary database to send the redo data to a new logical standby database in the corporate offices. Because the

systems used in the corporate office are different for the physical standby database and the proposed logical standby database, you cannot use the `DEPENDENCY` attribute when defining the standby destinations. Because redo data needs to be transmitted over a WAN, it would degrade performance on the primary database to send the redo data twice, which has been deemed to be unacceptable.

Cascaded destinations can solve this problem. To set this up, you would create a logical standby database following the instructions in [Chapter 4](#), but you would also set up the corporate physical standby database to transmit the redo data over the corporate LAN to the new logical standby database. In this way, the primary database is only sending the data once over the WAN to the physical standby database. The logical standby database could then be modified with new materialized views so that the marketing group can manage the data more efficiently. Because the logical standby database is opened and in read/write mode, the marketing group can add new schemas and load in sales data without affecting performance on the primary database, or the viability and current state of the physical standby database.

E.3.4 Consolidated Reporting with Cascaded Logical Standby Destinations

You have five Sales offices around the world, each with its own primary database. You would like to implement a failure protection strategy for all of them, as well as a way to get timely access to all data with minimal effect on each primary database.

To solve this problem, you would first implement a no-data-loss environment for each of the five offices by creating a physical standby database (with `LGWR` and `SYNC` attributes) local to each office. The physical standby databases could be on a LAN. Then, create a logical standby database from each of the five primary databases and locate the logical standby databases in your corporate office. However, instead of having redo transport services on each of the five primary databases send the redo data, you would configure each of the five standby databases to send the redo data to its logical standby database over the WAN. At one logical standby database (or all of them), you would define database links to each of the other logical standby databases and use them to access all of the sales data. If you decide that you do not need all of the information from each of the five primary databases, but only certain tables, you can use the `SKIP` routines to stop applying data that you do not need on each of the logical standby databases.

E.3.5 Temporary Use of Cascaded Destinations During Network Upgrades

You have a primary database that is currently protected only by nightly backups. You have been told that you must implement a major failure recovery strategy immediately. You have another system of the same hardware type in-house, but it does not have enough power to serve as a standby database for failover purposes, and it does not have enough disks for the entire database. The only other system available to you that is large enough to hold the entire database is too far away to be put on the LAN, and the WAN that connects to it is extremely slow. The deadline for implementing the strategy is well before any network upgrades can occur. Adding a destination (on the primary database) to send the redo data to the remote location would severely affect performance.

The interim solution to this problem would be to create a physical standby database on the remote system and create a distribution repository on the smaller local system. A distribution repository contains only the standby control file, standby redo log files, and the standby database archived redo log files, not the datafiles. You would configure the primary database to send the redo data to the repository locally using the log writer process (`LGWR`) in synchronous mode (`SYNC`). Because the connection is over the LAN, the effect on performance would be minimal. The repository would

then be configured to send the data onward over the WAN to the real standby database.

The risk with this configuration is that while the primary database has transmitted all of its data to a standby database, it is possible that the repository has not completed sending the data to the remote standby database at the time of a failure at the primary database. In this environment, as long as both systems do not fail at the same time, the remote standby database should receive all the data sent up to the last log switch. You would have to send the current redo log file manually.

Once the WAN is upgraded to permit a direct connection to the remote standby database, you can either redirect the destination to the repository to point to the remote standby database directly or create a new destination to the remote standby database and continue transmitting to the repository as an archive log repository.

Creating a Standby Database with Recovery Manager

This appendix describes how to use Oracle Recovery Manager to create a standby database. This appendix contains the following topics:

- [Preparing to Use RMAN to Create a Standby Database](#)
- [Creating a Standby Database with RMAN: Overview](#)
- [Setting Up the Standby Database](#)
- [Creating a Standby Database with the Same Directory Structure](#)
- [Creating a Standby Database with a Different Directory Structure](#)
- [Creating a Standby Database on the Local Host](#)
- [Creating a Standby Database with Image Copies](#)
- [Usage Scenario](#)

F.1 Preparing to Use RMAN to Create a Standby Database

There are several advantages to using RMAN to create a standby database:

- RMAN creates standby databases using backups of the primary database, restoring datafiles to the standby site from backups. Thus, the primary database is not affected during the creation of standby databases.
- RMAN automates renaming of files including Oracle Managed Files (OMF), and directory structures.
- RMAN restores archived redo log files from backups and performs recovery to catch up the standby database to the primary database.

The procedure for preparing a standby database with RMAN is basically the same as for preparing a duplicate database. Nevertheless, you need to amend the duplication procedures described in *Oracle Database Backup and Recovery Advanced User's Guide* to account for the issues specific to a standby database.

Familiarize yourself with how to create a standby database in [Chapter 3, "Creating a Physical Standby Database"](#) and [Chapter 4, "Creating a Logical Standby Database"](#) before you attempt the RMAN creation procedures described in this chapter.

This section contains these topics:

- [About Standby Database Preparation Using RMAN](#)
- [Creating the Standby Control File with RMAN](#)

- [Naming the Standby Database Datafiles When Using RMAN](#)
- [Naming the Standby Database Log Files When Using RMAN](#)

F.1.1 About Standby Database Preparation Using RMAN

You can use either manual methods or the Recovery Manager `DUPLICATE` command to create a standby database from backups of your primary database. Before you perform the creation procedure, you must prepare the standby instance. You can use RMAN to do the preparation tasks described in [Table F-1](#).

Table F-1 Standby Database Preparation Using RMAN

Task	Procedure
Make a backup of the primary database to use to create a standby database.	Use the normal backup procedure for your primary database as documented in <i>Oracle Database Backup and Recovery Basics</i> .
Create a backup of the primary control file that is usable as a standby control file (if you do not have one).	See Section F.1.2, "Creating the Standby Control File with RMAN" .
Choose file names for the standby datafiles.	See Section F.1.3, "Naming the Standby Database Datafiles When Using RMAN" .
Choose file names for the standby database archived redo log files and standby redo log files.	See Section F.1.4, "Naming the Standby Database Log Files When Using RMAN" .

In addition to the RMAN tasks described in [Table F-1](#), you must perform the following tasks to set up your standby database:

- Set all necessary initialization parameters on the primary database.
- Create an initialization parameter file for the standby database and configure all necessary parameters.
- Set up and configure Oracle Net, as required, to connect to the standby instance.
- Start the standby instance without mounting the control file.

See [Chapter 3](#) for a complete discussion of physical standby database preparation, including initialization parameter settings. You must perform all necessary preparation tasks described in these chapters before RMAN can successfully create the standby database files and mount the standby database.

F.1.2 Creating the Standby Control File with RMAN

You can create the standby control file using either the RMAN `BACKUP` or `COPY` commands by performing the following steps:

Step 1 Connect to the primary database.

Connect to the primary database and, if desired, the recovery catalog database. For example, enter:

```
% rman TARGET SYS/oracle@trgt CATALOG rman/cat@catdb
```

Step 2 Create the standby control file.

Use either of the following commands to create the standby control file. The only difference between `BACKUP` and `COPY` commands is that the file format of the backup file is different.

- Using the `BACKUP` command

Mount the primary database and create the standby control file with the `BACKUP CURRENT CONTROLFILE FOR STANDBY` command. The following example uses a configured channel to create the standby control file. Then open the database, archive all unarchived redo log files, and back up any log files that have not yet been backed up at least once:

```
STARTUP MOUNT
BACKUP CURRENT CONTROLFILE FOR STANDBY;
SQL> ALTER DATABASE OPEN;
SQL 'ALTER SYSTEM ARCHIVE LOG CURRENT'; # so backup is consistent and
recoverable
BACKUP ARCHIVELOG ALL NOT BACKED UP 1 TIMES;
```

- Using the `COPY` command

Copy the current primary control file. Specify the `FOR STANDBY` option of the `COPY CURRENT CONTROLFILE` command to make a copy of the current control file that is usable as a standby control file. For example:

```
COPY CURRENT CONTROLFILE FOR STANDBY TO '/tmp/sby_control01.ctl';
```

Step 1 List the backup sets or image copies.

If desired, issue a `LIST` command to see a listing of the backup sets and pieces, or issue a `LIST COPY` command to see a listing of the image copies.

Note: If you already created a standby control file with the `SQL ALTER DATABASE CREATE STANDBY CONTROLFILE AS` statement, you can use the `RMAN CATALOG` command to add metadata about the standby control file to the recovery catalog:

```
CATALOG CONTROLFILECOPY '/tmp/sby_control01.ctl';
```

F.1.3 Naming the Standby Database Datafiles When Using RMAN

A standby database can reside either on the same host as the primary database or on a different host. The following table illustrates the implications for renaming the standby database datafiles depending on if the directory structures on the hosts are the same or different.

Standby Database Host	Directory Structure	Renaming
Same host as primary	Different from primary host	Necessary.
Same host as primary	Same as primary host	Illegal. The standby database datafiles cannot exist in the same directories as the primary database datafiles on the same host.
Different host from primary	Same as primary host	Not necessary.
Different host from primary	Different from primary host	Necessary.

When the directory structures are *different* for the primary and standby hosts, you have these options for naming the standby datafiles:

- Configuring the standby database initialization parameter `DB_FILE_NAME_CONVERT`

- Use the `DB_FILE_NAME_CONVERT` option of the RMAN `DUPLICATE` command
- By using the RMAN `CONFIGURE AUXNAME` or `SET NEWNAME` command when creating the standby database

When the directory structures are the *same* for the primary and standby hosts, then you have these naming options:

- Leaving the standby file names the same as the primary file names (that is, not setting `DB_FILE_NAME_CONVERT` or issuing a `CONFIGURE AUXNAME` or `SET NEWNAME` command) and specifying the `NOFILENAMECHECK` option of the `DUPLICATE` command
- By using the `DB_FILE_NAME_CONVERT` parameter, or the `CONFIGURE AUXNAME` or `SET NEWNAME` commands to rename the standby datafiles

Note that when you use `DB_FILE_NAME_CONVERT`, the format is as follows:

```
DB_FILE_NAME_CONVERT = 'oldstring1', 'newstring1', 'oldstring2', 'newstring2', ...
```

For example, you can specify the `DB_FILE_NAME_CONVERT` initialization parameter as follows:

```
DB_FILE_NAME_CONVERT = '/dbs/t1/', '/dbs/t1/s_', '/dbs/t2/', '/dbs/t2/s_'
```

Because you can specify datafile file names in the standby control file in multiple ways, a method for prioritizing settings is necessary. [Table F-2](#) specifies the hierarchy for the naming of datafiles in the standby database.

Table F-2 Order of Precedence for Naming Datafiles in Standby Database

	Method of Standby Datafile Naming	Requirement
1	Issue <code>SET NEWNAME</code> command.	You must issue this command in the <code>RUN</code> block for the creation of the standby database.
2	Use the <code>DB_FILE_NAME_CONVERT</code> option of the RMAN <code>DUPLICATE</code> command.	None.
3	Issue <code>CONFIGURE AUXNAME</code> command.	You must be connected to a recovery catalog, and an <code>AUXNAME</code> that is not <code>NULL</code> must be stored in the catalog for the datafile.
4	Datafile file name as currently specified in the standby control file. The standby file name is identical to the primary file name or is named with the <code>DB_FILE_NAME_CONVERT</code> parameter.	If the file name is different, then the <code>DB_FILE_NAME_CONVERT</code> parameter must be set in the standby initialization parameter file. If the file name is the same, then you must specify the <code>NOFILENAMECHECK</code> clause of the <code>DUPLICATE</code> command.

See *Oracle Database Reference* for more information about how to use `DB_FILE_NAME_CONVERT` to name files on a standby database.

F.1.4 Naming the Standby Database Log Files When Using RMAN

Redo log files are not created on the standby database by RMAN. However, as described in [Chapter 3](#), log files can be created by other actions that you perform on the standby database. After the log files are created, they are maintained and archived according to the normal rules for log files.

The only option when naming the redo log files on the standby database is the file names for the log files, as specified in the standby control file. If the log file names on the standby must be different from the primary file names, then one option is to

specify file names for the redo logs by setting `LOG_FILE_NAME_CONVERT` in the standby initialization parameter file.

Note these restrictions when specifying file names for the redo log files on the standby database:

- You must use the `LOG_FILE_NAME_CONVERT` parameter to name the redo log files if the primary and standby databases use different naming conventions for the log files.
- You cannot use the `SET NEWNAME` or `CONFIGURE AUXNAME` commands to rename the redo log files.
- You cannot use the `LOGFILE` clause of the `DUPLICATE` command to specify file names for the redo log files.
- If you want the redo log file names on the standby database to be the same as the primary redo log file names, then you must specify the `NOFILENAMECHECK` clause of the `DUPLICATE` command. Otherwise, RMAN signals an error even if the standby database is created in a different host.

F.2 Creating a Standby Database with RMAN: Overview

When you create a standby database, the procedure differs depending on whether the standby database is on the same host as the primary database or on a different host. The procedures in this chapter assume that you have already completed the standby setup and preparation as outlined in [Chapter 3](#). Do not use these procedures until you have made all necessary initialization parameter settings and network configuration changes.

After you have performed the steps necessary for preparing the standby instance, issue the Recovery Manager `DUPLICATE . . . FOR STANDBY` command to create the standby database using backups of the primary database. Note that a standby database, unlike a duplicate database created by `DUPLICATE` *without* the `FOR STANDBY OPTION`, does not get a new DBID. Hence, you should not register the standby database with your recovery catalog.

The steps for creating the standby database differ depending on the following;

- Whether or not you specify that RMAN should recover the standby database after creating it
- Whether or not you use Oracle Managed Files (OMF)

See *Oracle Database Backup and Recovery Advanced User's Guide* to learn how to use the `DUPLICATE` command to create a duplicate database that is not a standby database.

F.2.1 RMAN Standby Creation Without Recovery

By default, RMAN does not recover the standby database after creating it. If you do not specify the `DORECOVER` option of the `DUPLICATE` command, then RMAN automates these steps of the standby creation procedure during duplication:

1. RMAN establishes connections both to the primary and standby databases, and the recovery catalog (if used).
2. RMAN queries the repository, which is either the primary control file or the recovery catalog, to identify the backups of primary database datafiles and the standby control file.

3. If you use a media manager, then RMAN contacts the media manager on the standby host to request the backup data.
4. RMAN restores the standby control file to the standby host, thereby creating the standby control file.
5. RMAN restores the primary datafile backups and copies to the standby host, thereby creating the standby database datafiles.
6. RMAN leaves the standby database mounted, but does *not* place the standby database in manual or managed recovery mode. RMAN disconnects and does not perform media recovery of the standby database. Note that you should *not* register the standby database in the recovery catalog.

F.2.2 RMAN Standby Creation with Recovery

If you do specify the `DORECOVER` option of the `DUPLICATE` command, then RMAN performs the same Steps 1-5 in [Section F.2.1](#). Instead of Step 6, it performs these steps:

1. After all data is restored, RMAN begins media recovery. If recovery requires archived redo log files, and if the log files are not already on disk, RMAN attempts to restore it from backups.
2. RMAN recovers the standby database to the specified time, system change number (SCN), or log file sequence number, or to the latest archived redo log file generated if none of the preceding are specified.
3. RMAN leaves the standby database mounted after media recovery is complete, but does *not* place the standby database in manual or managed recovery mode. Note that you should *not* register the standby database in the recovery catalog.

Note: After RMAN creates the standby database, you must resolve any gap sequence before placing it in manual or managed recovery mode, or opening it in read-only mode. [Section 5.8](#) discusses gap sequence resolution in detail.

If you want RMAN to recover the standby database after creating it, then the standby control file must be usable for the desired recovery. Thus, these conditions must be met:

- The end recovery time of the standby database must be greater than or equal to the checkpoint SCN of the standby control file.
- An archived redo log file containing the checkpoint SCN of the standby control file must be available at the standby site for recovery.

One way to ensure these conditions are met is to issue the `ALTER SYSTEM ARCHIVE LOG CURRENT` statement after creating the standby control file. This statement archives the online redo log files of the primary database. Then, either back up the most recent archived redo log file with RMAN or move the archived redo log file to the standby site.

Note: The procedures in this chapter assume that you are using RMAN backups to create the standby database. If you are using RMAN image copies, then refer to [Section F.7](#).

See *Oracle Database Backup and Recovery Reference* for the list of `DUPLICATE` restrictions for creating a standby database with RMAN.

F.3 Setting Up the Standby Database

No matter which standby creation scenario you choose, you must first start the standby database and then connect RMAN to this database. The details of this procedure vary depending on whether or not the standby and primary systems have a different directory structure, and whether or not Oracle Managed Files (OMF) is used:

- [Setting Up a Standby Database When Files Are Not Oracle Managed Files](#)
- [Setting Up a Standby Database When All Files Are Oracle Managed Files](#)
- [Setting Up a Standby Databases When a Subset of Files Are Oracle Managed Files](#)

F.3.1 Setting Up a Standby Database When Files Are Not Oracle Managed Files

Perform the following steps to set up a standby database that does not use OMF:

1. Use an operating system utility to copy the SPFILE (or the initialization parameter file) from the target host to the standby host. Set all required parameters in the standby database's initialization parameter file, as described in [Section 3.2.3](#).

For example, if creating the standby database on a separate host with a different directory structure, edit these initialization parameters:

- Those that end with `_DEST` and `_PATH` and specify a path name
- `DB_FILE_NAME_CONVERT` so that it captures *all* the target datafiles and converts them appropriately, for example, from `tbs_*` to `sbytbs_*`
- `LOG_FILE_NAME_CONVERT` so that it captures *all* the redo log files and converts them appropriately, for example, `log_*` to `sbylog_*`

For example, the following are sample parameter settings in the standby database initialization parameter file:

```
STANDBY_ARCHIVE_DEST = /fs3/arc_dest/
LOG_ARCHIVE_FORMAT = log%d_%t_%s_%r.arc
DB_FILE_NAME_CONVERT = '/oracle', '/fs3/oracle', '/dbf', '/fs3/oracle'
LOG_FILE_NAME_CONVERT = '/oracle', '/fs3/oracle'
```

2. Use SQL*Plus to start the standby instance without mounting it. For example, enter the following to connect to `sbdb1` as `SYS` (who has `SYSDBA` privileges) and start the database:

```
SQL> CONNECT SYS/sys_pwd@sbdb1 AS SYSDBA
SQL> STARTUP NOMOUNT PFILE=initSBDB1.ora
```

3. Use SQL*Plus to mount or open the primary database if it is not already mounted or open. For example, enter the following to connect to `prod1` as `SYS` and open the database:

```
SQL> CONNECT SYS/sys_pwd@prod1 AS SYSDBA
SQL> STARTUP PFILE=initPROD1.ora
```

Ensure the recovery catalog database is open. For example, enter the following to connect to `catdb` as `SYS` and open the recovery catalog database:

```
SQL> CONNECT SYS/oracle@catdb AS SYSDBA
SQL> STARTUP PFILE=initCATDB.ora
```

4. The standby instance must be accessible through Oracle Net. Before proceeding, use SQL*Plus to ensure you can establish a connection to the standby instance. Note that you must connect to the standby instance with SYSDBA privileges, so a password file must exist.
5. Connect to the target database, the standby instance, and (if you use one) the recovery catalog database. Note that you specify the *primary* database with the TARGET keyword and the *standby* instance with the AUXILIARY keyword.

In the following example, connection is established without a recovery catalog by using operating system authentication:

```
% rman TARGET / AUXILIARY SYS/sys_pwd@sbdb1
```

F.3.2 Setting Up a Standby Database When All Files Are Oracle Managed Files

If the standby database uses OMF for all files, you must set the following parameters on the standby database or auxiliary instance before you perform the RMAN duplication. Oracle recommends using a server parameter file (SPFILE) when the database control files are OMF files. Otherwise, at end of the following procedure, be sure to update the CONTROL_FILE initialization parameter with the value of the control file names column in V\$CONTROLFILE view.

Required initialization parameter changes:

1. Set up a different DB_NAME parameter for duplicate databases and a different DB_UNIQUE_NAME parameter for standby databases
2. Do not set the LOG_FILE_NAME_CONVERT and DB_FILE_NAME_CONVERT parameters
3. Do not set the CONTROL_FILE parameter (if the database control files are to be OMF files)
4. Set the DB_CREATE_FILE_DEST parameter (all datafiles, control file and online redo log will be created in this destination)
5. Set the STANDBY_FILE_MANAGEMENT=AUTO parameter (for standby database duplication only)

You can use the ALTER SYSTEM RESET statement to clear initialization parameters in the SPFILE for steps 2 and 3. For example:

```
ALTER SYSTEM RESET CONTROL_FILE SCOPE=SPFILE SID='*'
```

Optional initialization parameter changes:

Set the DB_CREATE_FILE_DEST parameter to create one copy of the control file and one member of the online redo log. To create more than one control file and online redo log file, set the DB_CREATE_ONLINE_LOG_DEST_n (where *n* can be an integer from 1 to 5) parameter, depending on the number of multiplexed copies you require. The control file and online redo log files are not created in DB_CREATE_FILE_DEST when one DB_CREATE_ONLINE_LOG_DEST_n parameter is set. For example, if you set DB_CREATE_ONLINE_LOG_DEST_1 and DB_CREATE_ONLINE_LOG_DEST_2, then you will get two copies of the control file and online redo log file on each destination.

F.3.3 Setting Up a Standby Databases When a Subset of Files Are Oracle Managed Files

If only some of the primary database files are OMF files, or if there are datafiles spread across more than one disk group, you may want to create a duplicate database with exactly the same structure.

For example, fast datafiles may reside in '+FAST' disk group and slow datafiles may reside in '+SLOW' disk group. After duplicating the database, you would like to have its datafiles reside in '+FAST2' disk group and its slow datafiles reside in '+SLOW2' disk group. To set up the standby database, configure the LOG_FILE_NAME_CONVERT and DB_FILE_NAME_CONVERT parameters as ('+FAST', '+FAST2', '+SLOW', '+SLOW2'). If you want the database to be in the same disk group but with a different name, then change the middle part of the name. For example, use ('/boston/', '/sfo/') where boston is the DB_UNIQUE_NAME of the primary database and sfo is the DB_UNIQUE_NAME of the duplicate database.

Use the following steps to set initialization parameters on the standby database or auxiliary instance before performing the RMAN duplication.

Required initialization parameter changes:

1. Set up a different DB_NAME parameter for duplicate database and a different DB_UNIQUE_NAME for standby database
2. Set up the LOG_FILE_NAME_CONVERT and DB_FILE_NAME_CONVERT parameters to convert file names
3. Modify the CONTROL_FILE parameter to specify new control files

F.4 Creating a Standby Database with the Same Directory Structure

The simplest case is to create the standby database on a different host and to use the same directory structure. In this case, you do *not* need to set the DB_FILE_NAME_CONVERT or LOG_FILE_NAME_CONVERT parameters in the standby initialization parameter file or set new file names for the standby datafiles. The primary and standby datafiles and log files have the same file names.

F.4.1 Creating the Standby Database Without Performing Recovery

To create the standby database without performing recovery, do not specify the DORECOVER option on the DUPLICATE command. By default, RMAN leaves the standby database mounted and does not recover it.

To create a standby database without performing recovery:

1. Follow the steps in [Section F.3, "Setting Up the Standby Database"](#). Make sure to set all necessary parameters in the standby initialization parameter file.
2. Follow these steps during duplication to create but not recover the standby datafiles:
 - a. If automatic channels are not configured, then allocate at least one auxiliary channel. This channel performs the work of duplication.
 - b. Specify NOFILENAMECHECK in the DUPLICATE command. The NOFILENAMECHECK option is required when the standby and primary datafiles and log files have the same names. Otherwise, RMAN returns an error.

For example, run the following command to create the standby database:

```
DUPLICATE TARGET DATABASE FOR STANDBY
NOFILENAMECHECK;
```

F.4.2 Creating the Standby Database and Performing Recovery

To create the standby database and perform recovery, specify the `DORECOVER` option on the `DUPLICATE` command.

To create a standby database and perform recovery:

1. Follow the steps in [Section F.3](#). Make sure to set all necessary parameters in the standby initialization parameter file.
2. Follow these steps to restore and recover the standby datafiles:
 - a. Ensure the end recovery time is greater than or equal to the checkpoint SCN of the standby control file and that a log file containing the checkpoint SCN is available for recovery.
 - b. If desired, issue a `SET` command to specify the end time, SCN, or log sequence number for incomplete recovery.
 - c. If automatic channels are not configured, then manually allocate at least one auxiliary channel.
 - d. Specify the `NOFILENAMECHECK` parameter in the `DUPLICATE` command, and use the `DORECOVER` option.

For example, enter the following at the RMAN prompt to use a configured channel to create the standby database:

```
# If desired, issue a LIST command to determine the SCN of the standby control
file.
# The SCN to which you recover must be greater than or equal to the standby
control
# file SCN.
LIST BACKUP OF CONTROLFILE;
LIST COPY OF CONTROLFILE;

RUN
{
  # If desired, issue a SET command to terminate recovery at a specified point.
  # SET UNTIL SCN 143508;
  DUPLICATE TARGET DATABASE FOR STANDBY
  NOFILENAMECHECK
  DORECOVER;
}
```

RMAN uses all incremental backups, archived redo log file backups, and archived redo log files to perform incomplete recovery. The standby database is left mounted.

F.5 Creating a Standby Database with a Different Directory Structure

If you create the standby database on a host with a different directory structure, you need to specify new file names for the standby database datafiles and redo log files. You can do the following:

- Set the `LOG_FILE_NAME_CONVERT` parameter in the standby initialization parameter file to name the redo log files on the standby database. If you do not set `LOG_FILE_NAME_CONVERT`, then you must use the `NOFILENAMECHECK` option of the `DUPLICATE` command.

- Set the `DB_FILE_NAME_CONVERT` parameter in the standby initialization parameter file to name the standby datafiles.
- Issue the `SET NEWNAME` command or the `CONFIGURE AUXNAME` command when using the `RMAN DUPLICATE` command to name the datafiles.

When creating the standby database on a host with a different directory structure, follow one of the procedures in the following sections:

- [Naming Standby Database Files with `DB_FILE_NAME_CONVERT`](#)
- [Naming Standby Database Files with `SET NEWNAME`](#)
- [Naming Standby Database Files with `CONFIGURE AUXNAME`](#)

See *Oracle Database Backup and Recovery Advanced User's Guide* to learn about the difference between `SET NEWNAME` and `CONFIGURE AUXNAME`, and [Chapter 3](#) for a complete discussion of physical standby database preparation and creation.

F.5.1 Naming Standby Database Files with `DB_FILE_NAME_CONVERT`

In this procedure, you use the `DB_FILE_NAME_CONVERT` parameter to name the standby datafiles and the `LOG_FILE_NAME_CONVERT` parameter to name the redo log files on the standby database. See [Section 3.1.4](#) for examples of how to use the `DB_FILE_NAME_CONVERT` and `LOG_FILE_NAME_CONVERT` parameters to name standby database files.

F.5.1.1 Creating the Standby Database Without Performing Recovery

To create the standby database without performing recovery, do not specify the `DORECOVER` option on the `DUPLICATE` command. By default, RMAN leaves the standby database mounted and does not recover it.

To use parameters to name standby files without performing recovery:

1. Follow the steps in [Section F.3](#). Make sure to set all necessary parameters in the standby initialization parameter file.
2. Run the `DUPLICATE` command. For example, run the following:

```
DUPLICATE TARGET DATABASE FOR STANDBY;
```

After restoring the backups, RMAN leaves the standby database mounted.

F.5.1.2 Creating the Standby Database and Performing Recovery

After using the `DB_FILE_NAME_CONVERT` parameter to name the standby datafiles and the `LOG_FILE_NAME_CONVERT` parameter to name the log files on the standby database, specify the `DORECOVER` option on the `DUPLICATE` command to create the standby database and perform recovery. The steps in the procedure are the same as for [Section F.4.2](#).

F.5.2 Naming Standby Database Files with `SET NEWNAME`

In this procedure, you use `SET NEWNAME` commands to name the standby datafiles.

F.5.2.1 Creating the Standby Database Without Performing Recovery

To create the standby database without performing recovery, do not specify the `DORECOVER` option on the `DUPLICATE` command. By default, RMAN leaves the standby database mounted and does not recover it.

To name standby database files with the SET NEWNAME command without performing recovery:

1. Follow the steps in [Section F.3](#). Make sure to set all necessary parameters in the standby initialization parameter file.
2. Run the DUPLICATE command. Perform the following steps:
 - a. If automatic channels are not configured, then manually allocate at least one auxiliary channel.
 - b. Specify new file names for the standby database datafiles with SET NEWNAME commands.
 - c. Issue the DUPLICATE command.

The following example uses a configured channel to create the standby database:

```

RUN
{
  # set new file names for the datafiles
  SET NEWNAME FOR DATAFILE 1 TO '?/dbs/standby_data_01.f';
  SET NEWNAME FOR DATAFILE 2 TO '?/dbs/standby_data_02.f';
  .
  .
  .
  # run the DUPLICATE command
  DUPLICATE TARGET DATABASE FOR STANDBY;
}

```

F.5.2.2 Creating the Standby Database and Performing Recovery

To create the standby database and perform recovery, specify the DORECOVER option on the DUPLICATE command.

To use the SET NEWNAME command to name standby database files and perform recovery:

1. Follow the steps in [Section F.3](#). Make sure to set all necessary parameters in the standby initialization parameter file.
2. Run the DUPLICATE command. Follow these steps:
 - a. Ensure the end recovery time is greater than or equal to the checkpoint SCN of the standby control file and that a log file containing the checkpoint SCN is available for recovery (as described in [Section F.2.2](#)).
 - b. If desired, issue a SET command to specify the end time, SCN, or log sequence number for incomplete recovery.
 - c. If automatic channels are not configured, then manually allocate at least one auxiliary channel.
 - d. Specify new file names for the standby database datafiles.
 - e. Issue the DUPLICATE command with the DORECOVER option.

For example, enter the following at the RMAN prompt to use a configured channel to create the standby database:

```

# If desired, issue a LIST command to determine the SCN of the standby control
file.
# The SCN to which you recover must be greater than or equal to the control
file SCN.

LIST BACKUP OF CONTROLFILE;

```

```

LIST COPY OF CONTROLFILE;
RUN
{
  # If desired, issue a SET command to terminate recovery at a specified point.
  # SET UNTIL TIME 'SYSDATE-7';

  # Set new file names for the datafiles
  SET NEWNAME FOR DATAFILE 1 TO '?/dbs/standby_data_01.f';
  SET NEWNAME FOR DATAFILE 2 TO '?/dbs/standby_data_02.f';
  .
  .
  .
  DUPLICATE TARGET DATABASE FOR STANDBY
    DORECOVER;
}

```

RMAN uses all incremental backups, archived redo log file backups, and archived redo log files to perform incomplete recovery. The standby database is left mounted.

F.5.3 Naming Standby Database Files with CONFIGURE AUXNAME

In this procedure, you use CONFIGURE AUXNAME commands to name the standby datafiles.

F.5.3.1 Creating the Standby Database Without Performing Recovery

To create the standby database without performing recovery, do not specify the DORECOVER option on the DUPLICATE command. By default, RMAN leaves the standby database mounted and does not recover it.

To use CONFIGURE AUXNAME to name standby database files without performing recovery:

1. Follow the steps in [Section F.3](#). Make sure to set all necessary parameters in the standby initialization parameter file.
2. Configure the auxiliary names for the datafiles. For example, enter:

```

# set auxiliary names for the datafiles
CONFIGURE AUXNAME FOR DATAFILE 1 TO '/oracle/auxfiles/aux_1.f';
CONFIGURE AUXNAME FOR DATAFILE 2 TO '/oracle/auxfiles/aux_2.f';
.
.
.
CONFIGURE AUXNAME FOR DATAFILE n TO '/oracle/auxfiles/aux_n.f';

```

3. Run the DUPLICATE command. If automatic channels are not configured, manually allocate at least one auxiliary channel before issuing the DUPLICATE command, as in the following example:

```

RUN
{
  # allocate at least one auxiliary channel of type DISK or sbt
  ALLOCATE AUXILIARY CHANNEL standby1 DEVICE TYPE sbt;
  .
  .
  .
  # issue the DUPLICATE command
  DUPLICATE TARGET DATABASE FOR STANDBY;
}

```

4. Unspecify the auxiliary names for the datafiles so that they are not overwritten by mistake. For example, enter the following at the RMAN prompt:

```
# un-specify auxiliary names for the datafiles
CONFIGURE AUXNAME FOR DATAFILE 1 CLEAR;
CONFIGURE AUXNAME FOR DATAFILE 2 CLEAR;
.
.
.
CONFIGURE AUXNAME FOR DATAFILE n CLEAR;
```

F.5.3.2 Creating the Standby Database and Performing Recovery

To create the standby database and perform recovery, specify the `DORECOVER` option on the `DUPLICATE` command.

To use `CONFIGURE AUXNAME` to name standby files and perform recovery:

1. Follow the steps in [Section F.3](#). Make sure to set all necessary parameters in the standby initialization parameter file.
2. Set the auxiliary names for the datafiles. For example, enter the following:

```
# set auxiliary names for the datafiles
CONFIGURE AUXNAME FOR DATAFILE 1 TO '/oracle/auxfiles/aux_1.f';
CONFIGURE AUXNAME FOR DATAFILE 2 TO '/oracle/auxfiles/aux_2.f';
.
.
.
CONFIGURE AUXNAME FOR DATAFILE n TO '/oracle/auxfiles/aux_n.f';
```

3. Run the `DUPLICATE` command. Follow these steps:
 - Ensure the end recovery time is greater than or equal to the checkpoint SCN of the standby control file and that a log file containing the checkpoint SCN is available for recovery (as described in [Section F.2.2](#)).
 - If desired, issue a `SET` command to specify the end time, SCN, or log sequence number for incomplete recovery.
 - If automatic channels are not configured, then manually allocate at least one auxiliary channel.
 - Issue the `DUPLICATE TARGET DATABASE` for standby command.

For example, enter the following at the RMAN prompt to use a configured channel to create the standby database:

```
# If desired, issue a LIST command to determine the SCN of the standby control
file.
# The SCN to which you recover must be greater than or equal to the control
file SCN.
LIST BACKUP OF CONTROLFILE;
LIST COPY OF CONTROLFILE;

DUPLICATE TARGET DATABASE FOR STANDBY
DORECOVER;
```

RMAN uses all incremental backups, archived redo log file backups, and archived redo log files to perform incomplete recovery. The standby database is left mounted.

4. Clear the auxiliary name settings for the datafiles so that they are not overwritten by mistake. For example, enter the following at the RMAN prompt:

```
# un-specify auxiliary names for the datafiles
CONFIGURE AUXNAME FOR DATAFILE 1 CLEAR;
CONFIGURE AUXNAME FOR DATAFILE 2 CLEAR;
.
.
.
CONFIGURE AUXNAME FOR DATAFILE n CLEAR;
```

F.6 Creating a Standby Database on the Local Host

When creating a standby database on the same host as the primary database, follow the same procedure as for duplicating to a remote host with a different directory structure as described in [Section F.5](#).

Note the following restrictions when creating a standby database on the same host as the primary database:

- You can create the standby database in the same Oracle home as the target database, but you must convert the file names with the same methods used for conversion on a separate host. That is, you must treat a standby database in the same Oracle home as if it were a database on a separate host with a different directory structure. You must *not* use the same names for standby and primary database files when the two databases are on the same machine.
- You must set the `DB_UNIQUE_NAME` initialization parameter on both databases.

CAUTION: Do not use the `NOFILENAMECHECK` option when creating the standby database in the same Oracle home as the primary database. If you do, then you may overwrite the target database files or cause the `DUPLICATE` command to fail with an error.

F.7 Creating a Standby Database with Image Copies

This section contains these topics:

- [Overview](#)
- [When Copies and Datafiles Use the Same Names](#)
- [When Copies and Datafiles Use Different Names](#)

F.7.1 Overview

The main restriction when using RMAN image copies to create the standby datafiles is that the image copy file names for datafiles and archived redo log files on the primary and standby hosts must be the same. For example, assume that datafile 1 is named `/oracle/dbs/df1.f` on the primary host. If you use the RMAN `COPY` command to copy this datafile to `/data/df1.f`, then this image copy must exist on the standby host with the same file name of `/data/df1.f`. Otherwise, RMAN cannot locate the metadata for the standby image copy in its repository.

You have two main ways of populating the standby host with the image copies:

- Transferring them manually with `ftp` or some other utility

- Mounting the standby directory structure on the primary host with a network file system (NFS)

When you use the NFS method, you can create a directory on the primary host that maps to a directory on the standby host. If you use this method, then the NFS mount point on both machines must have the same directory name. For example, you can map `/data` on the primary host to `/data` on the standby host, but you cannot map `/data` on the primary host to `/dir` on the standby host (unless you use functionality such as symbolic links in UNIX or logical drives on Windows).

The file name of the image copy on the standby host must be the same as the file name of the image copy on the primary host. Nevertheless, you can specify a different path name for the standby datafile by using `SET NEWNAME` commands or the `DB_FILE_NAME_CONVERT` initialization parameter.

For example, although the image copy of datafile 1 is named `/data/df1.f` on the standby host, you can specify the path name `/oracle/sb/df1.f` in the standby control file by using initialization parameters or `RMAN` commands. Note that you do not manually rename the physical image copy. When you run the `DUPLICATE` command, `RMAN` restores the image copy `/data/df1.f` and creates the standby datafile 1 as `/oracle/sb/df1.f` based on the information in the initialization parameters or `RMAN` commands.

[Table F-3](#) illustrates two scenarios for using NFS to create a standby database with one datafile.

Table F-3 Using Image Copies to Create a Standby Database: Scenario

NFS Mount Point	Primary Datafile File Name	Image Copy File Name	Standby Datafile File Name	Procedure
<code>/data</code> (same on both hosts)	<code>/oracle/dbs/df1.f</code>	<code>/data/df1.f</code>	<code>/data/df1.f</code> (same path name as image copy)	See Section F.7.2, "When Copies and Datafiles Use the Same Names"
<code>/data</code> (same on both hosts)	<code>/oracle/dbs/df1.f</code>	<code>/data/df1.f</code>	<code>/oracle/sb/df1.f</code> (different path name from image copy)	See Section F.7.3, "When Copies and Datafiles Use Different Names"

[Table F-3](#) assumes that the standby directory structure is mounted on the primary host, and that the mount point is `/data` on both hosts. Because the primary host mounts the standby host directory structure, when you create the image copy `/data/df1.f` on the primary host, you are actually creating the image copy `/data/df1.f` on the standby host.

In the first scenario, you name the standby datafiles with the same file names as the image copies. This case is the simplest because you do not need to use `RMAN` at all to create the standby database. First, set the `DB_FILE_NAME_CONVERT` parameter in the standby initialization parameter file to convert the primary datafile file name `/oracle/dbs/df1.f` to the standby file name `/data/df1.f`. Then, copy the files to the standby host, and mount the standby database.

In the second scenario, you use different file names for the standby datafiles and the image copies. To create this standby database, run the `DUPLICATE` command. The `DUPLICATE` command restores the image copy of datafile 1 and renames it according to either the `SET NEWNAME` commands or the `DB_FILE_NAME_CONVERT` initialization parameter.

F.7.2 When Copies and Datafiles Use the Same Names

This procedure assumes that you are using the same file names for the standby datafiles and the image copies of the primary datafiles.

To create a standby database when the copies and standby datafiles have the same names:

1. After connecting to the primary database, and if desired, the recovery catalog database, mount but do not open the primary database and ensure the database was closed cleanly before mounting. For example, enter:

```
RMAN> STARTUP MOUNT PFILE=init.ora;
```

2. Make sure that you set `DB_FILE_NAME_CONVERT` in the standby initialization parameter file so that standby datafile file names are translated from the primary datafile file names. For example:

```
DB_FILE_NAME_CONVERT = '/oracle/dbs', '/dsk2/oracle'
```

3. Copy all of the datafiles and the standby control file. For example, enter:

```
COPY
  DATAFILE 1 TO '/dsk2/oracle/df_1.f',
  DATAFILE 2 TO '/dsk2/oracle/df_2.f',
  DATAFILE 3 TO '/dsk2/oracle/df_3.f',
  DATAFILE 4 TO '/dsk2/oracle/df_4.f',
  DATAFILE 5 TO '/dsk2/oracle/df_5.f',
  DATAFILE 6 TO '/dsk2/oracle/df_6.f',
  DATAFILE 7 TO '/dsk2/oracle/df_7.f',
  DATAFILE 8 TO '/dsk2/oracle/df_8.f',
  DATAFILE 9 TO '/dsk2/oracle/df_9.f',
  DATAFILE 10 TO '/dsk2/oracle/df_10.f',
  DATAFILE 11 TO '/dsk2/oracle/df_11.f',
  DATAFILE 12 TO '/dsk2/oracle/df_12.f',
  CURRENT CONTROLFILE FOR STANDBY TO '/dsk2/oracle/cf.f';
```

4. Start the standby instance and mount the standby control file. For example, start `SQL*Plus` and enter:

```
SQL> STARTUP NOMOUNT PFILE=/dsk2/oracle/dbs/initSTANDBY1.ora
SQL> ALTER DATABASE MOUNT;
```

F.7.3 When Copies and Datafiles Use Different Names

This procedure assumes that you use different file names for the standby datafiles and the image copies of the primary datafiles.

F.7.3.1 Creating the Standby Database Without Performing Recovery

To create the standby database without performing recovery, you do not need to run the `DUPLICATE` command. By default, `RMAN` leaves the standby database mounted and does not recover it.

To create a standby database when the copies and standby datafiles have different names without performing recovery:

1. Connect to the primary database, standby instance, and, if desired, the recovery catalog database. For example, enter:

```
% rman TARGET sys/sys_pwd@prod1 AUXILIARY sys/sys_pwd@sbdb1 CATALOG
rman/cat@catdb
```

2. Mount but do not open the primary database and ensure the database was closed cleanly before mounting. For example, enter:

```
STARTUP MOUNT PFILE=initPROD1.ora
```

3. Either set the DB_FILE_NAME_CONVERT initialization parameter on the standby database so that standby datafile file names are translated from the primary datafile file names, or issue SET NEWNAME commands. For example, set the DB_FILE_NAME_CONVERT parameter as follows:

```
DB_FILE_NAME_CONVERT = '/oracle/dbs', '/dsk2/oracle'
```

4. Use the COPY command to copy all of the datafiles and the standby control file. For example, issue the following commands:

```
COPY
  DATAFILE 1 TO '/dsk2/oracle/df_1.f',
  DATAFILE 2 TO '/dsk2/oracle/df_2.f',
  DATAFILE 3 TO '/dsk2/oracle/df_3.f',
  DATAFILE 4 TO '/dsk2/oracle/df_4.f',
  DATAFILE 5 TO '/dsk2/oracle/df_5.f',
  DATAFILE 6 TO '/dsk2/oracle/df_6.f',
  DATAFILE 7 TO '/dsk2/oracle/df_7.f',
  DATAFILE 8 TO '/dsk2/oracle/df_8.f',
  DATAFILE 9 TO '/dsk2/oracle/df_9.f',
  DATAFILE 10 TO '/dsk2/oracle/df_10.f',
  DATAFILE 11 TO '/dsk2/oracle/df_11.f',
  DATAFILE 12 TO '/dsk2/oracle/df_12.f',
  CURRENT CONTROLFILE FOR STANDBY TO '/dsk2/oracle/cf.f';
# To ensure the control file checkpoint is archived, archive the
# current redo log file
SQL 'ALTER SYSTEM ARCHIVE LOG CURRENT';
```

5. Start the auxiliary instance and mount the standby control file. For example, start SQL*Plus and enter:

```
SQL> STARTUP MOUNT PFILE=/dsk2/oracle/dbs/initSTANDBY1.ora
```

F.7.3.2 Creating the Standby Database and Performing Recovery

To create the standby database and perform recovery, specify the DORECOVER option on the DUPLICATE command.

To create a standby database when the copies and standby datafiles have different names and perform recovery:

1. Connect to the primary database, standby instance, and, if desired, the recovery catalog database. For example, enter:

```
% rman TARGET sys/sys_pwd@prod1 AUXILIARY sys/sys_pwd@sbdb1 CATALOG
rman/cat@catdb
```

2. Mount but do not open the primary database and ensure the database was closed cleanly before mounting. For example, enter:

```
STARTUP MOUNT PFILE=initPROD1.ora
```

3. Either set DB_FILE_NAME_CONVERT in the standby initialization parameter file so that standby datafile file names are translated from the primary datafile file names, or issue SET NEWNAME commands. For example, set the DB_FILE_NAME_CONVERT parameter as follows:

```
DB_FILE_NAME_CONVERT = '/oracle/dbs', '/dsk2/oracle'
```

4. Run the DUPLICATE command. Follow these steps:
 - a. Ensure the end recovery time is greater than or equal to the checkpoint SCN of the standby control file and that a log file containing the checkpoint SCN is available for recovery (as described in [Section F.2.2](#)).
 - b. If desired, issue a SET command to specify the end time, SCN, or log sequence number for recovery.
 - c. If automatic channels are not configured, then manually allocate at least one auxiliary channel for the duplication.
 - d. Copy every datafile and the standby control file.
 - e. Archive the current online redo log files.
 - f. Issue the DUPLICATE command with the DORECOVER option.

For example, enter the following:

```

COPY
  DATAFILE 1 TO '/dsk2/oracle/df_1.f',
  DATAFILE 2 TO '/dsk2/oracle/df_2.f',
  DATAFILE 3 TO '/dsk2/oracle/df_3.f',
  DATAFILE 4 TO '/dsk2/oracle/df_4.f',
  DATAFILE 5 TO '/dsk2/oracle/df_5.f',
  DATAFILE 6 TO '/dsk2/oracle/df_6.f',
  DATAFILE 7 TO '/dsk2/oracle/df_7.f',
  DATAFILE 8 TO '/dsk2/oracle/df_8.f',
  DATAFILE 9 TO '/dsk2/oracle/df_9.f',
  DATAFILE 10 TO '/dsk2/oracle/df_10.f',
  DATAFILE 11 TO '/dsk2/oracle/df_11.f',
  DATAFILE 12 TO '/dsk2/oracle/df_12.f',
  CURRENT CONTROLFILE FOR STANDBY TO '/dsk2/oracle/cf.f';
SQL 'ALTER SYSTEM ARCHIVE LOG CURRENT';
DUPLICATE TARGET DATABASE FOR STANDBY
  DORECOVER;

```

RMAN uses all incremental backups, archived redo log file backups, and archived redo log files to perform incomplete recovery. The standby database is left mounted.

F.8 Usage Scenario

In this scenario, you are performing a duplication that uses both backups and image copies of the primary datafiles. The scenario illustrates how RMAN is able to use both datafile backups and datafile copies for the standby files, and also is able to use both incremental backups and archived redo log files to recover the standby database.

Assume the following about the standby database environment:

- The primary database is on `host1` and the standby database is on `host2`.
- Database `prod1` has 30 datafiles: datafiles 1 through 25 are on a raw disk named with the pattern `/dev/rdsk###` (where `###` is a number starting with 001 and ending with 025), and datafiles 26 through 30 are located in the `/primary/datafile` directory.

You perform the following actions over the course of a week:

1. On Monday, you run the following incremental level 0 database backup:

```
BACKUP DEVICE TYPE sbt INCREMENTAL LEVEL 0 DATABASE PLUS ARCHIVELOG;
```

2. On Tuesday, copy datafiles 1 through 5 into the `/standby/datafile` directory on `host1`, then run the `BACKUP ARCHIVELOG ALL` command.
3. On Wednesday, copy datafiles 6 through 9 into the `/standby/datafile` directory on `host1`, then run the `BACKUP ARCHIVELOG ALL` command.
4. On Thursday, run the following incremental level 1 database backup:

```
BACKUP DEVICE TYPE sbt INCREMENTAL LEVEL 1 DATABASE PLUS ARCHIVELOG;
```

5. On Friday, copy datafiles 10 through 15 into the `/standby/datafile` directory on `host1`, then run the `BACKUP ARCHIVELOG ALL` command.
6. On Saturday morning, run the following RMAN commands:

```
COPY CURRENT CONTROLFILE FOR STANDBY TO '/standby/datafile/cf.f';
SQL 'ALTER SYSTEM ARCHIVELOG CURRENT';
BACKUP DEVICE TYPE sbt ARCHIVELOG ALL;
```

7. On Saturday night, ftp all the image copies in `/standby/datafile` on `host1` to `/standby/datafile` on `host2`, and also ftp all the log files on `host1` to `host2`; also make the tape backups of `prod1` accessible to `host2`.

On Sunday, you decide to create the standby database and recover it up to the point of the Saturday backup. You want all the standby datafiles to be located in the `/standby/datafile` directory on `host2`.

You must choose a method for naming the standby datafiles. You could use the `DB_FILE_NAME_CONVERT` parameter to change each pattern of the raw disk datafiles, which would require 25 pairs of values in the parameter (one pair for each raw disk file name that is being renamed). Instead, you decide to use `SET NEWNAME` commands for the 25 datafiles on raw disk, and use the `DB_FILE_NAME_CONVERT` parameter only for converting the names for the five datafiles in `/primary/datafile` to `/standby/datafile`.

The image copies are located in `/standby/datafile` on `host2`, but you only made copies of datafiles 1 through 15. This is not a problem, however, because you have incremental backups of all the datafiles. RMAN always chooses to restore image copies over backups, but if no image copies are available, then RMAN restores backups. So, you run the following script:

```
RUN
{
  # run SET NEWNAME commands for datafiles 1-25
  SET NEWNAME FOR DATAFILE 1 TO '/standby/datafile/df1.f';
  SET NEWNAME FOR DATAFILE 2 TO '/standby/datafile/df2.f';
  .
  .
  .
  SET NEWNAME FOR DATAFILE 25 TO '/standby/datafile/df25.f';
  DUPLICATE TARGET DATABASE FOR STANDBY DORECOVER;
}
```

RMAN does the following actions during the duplication:

- Uses the image copies of datafiles 1 through 15.
- Restores the backups of datafiles 16 through 30 (because no image copies are available of these datafiles).

- Uses incremental backups to recover datafiles 1 through 9 and datafiles 16 through 30, but not to recover datafiles 10 through 15 because these copies were created on Friday after the Thursday incremental level 1 backup.
- Restores and applies archived redo log files as needed to datafiles 1 through 30 up to the last archived redo log file that was backed up.
- Applies archived redo log files on disk up to the last archived redo log file.

Setting Archive Tracing

The Oracle database writes an audit trail of the archived redo log files received from the primary database into a trace file. The `LOG_ARCHIVE_TRACE` parameter controls output generated by the `ARCn`, `LGWR`, and foreground processes on the primary database, and the `RFS` and `FAL` server processes on the standby database.

G.1 LOG_ARCHIVE_TRACE Initialization Parameter

To see the archiving to the standby site, set the `LOG_ARCHIVE_TRACE` parameter in the primary and standby initialization parameter files. When you set the `LOG_ARCHIVE_TRACE` parameter, it causes the Oracle database to write an audit trail to a trace file as follows:

- On the primary database

This causes the Oracle database to write an audit trail of archiving process activity (`ARCn` and foreground processes, `LGWR`, and `FAL` activities) on the primary database in a trace file whose filename is specified in the `USER_DUMP_DEST` initialization parameter.
- On the standby database

This causes the Oracle database to write an audit trail of the `RFS` process and the `ARCn` process activity relating to archived redo log files on the standby database in a trace file whose filename is specified in the `USER_DUMP_DEST` initialization parameter.

G.2 Determining the Location of the Trace Files

The trace files for a database are located in the directory specified by the `USER_DUMP_DEST` parameter in the initialization parameter file. Connect to the primary and standby instances using `SQL*Plus`, and issue a `SHOW` statement to determine the location, for example:

```
SQL> SHOW PARAMETER USER_DUMP_DEST
NAME                                TYPE        VALUE
-----
user_dump_dest                       string      ?/rdbs/log
```

G.2.1 Setting the LOG_ARCHIVE_TRACE Initialization Parameter

The format for the archiving trace parameter is as follows, where *trace_level* is an integer:

```
LOG_ARCHIVE_TRACE=trace_level
```

To enable, disable, or modify the `LOG_ARCHIVE_TRACE` parameter for a physical standby database, issue a SQL statement similar to the following:

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_TRACE=15;
```

In the previous example, setting the `LOG_ARCHIVE_TRACE` parameter to a value of 15 sets trace levels 1, 2, 4, and 8 as described in [Section G.2.2](#).

Issue the `ALTER SYSTEM` statement from a different standby session so that it affects trace output generated by the remote file service (RFS) and `ARCn` processes when the next archived redo log file is received from the primary database. For example, enter:

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_TRACE=32;
```

G.2.2 Choosing an Integer Value

The integer values for the `LOG_ARCHIVE_TRACE` parameter represent levels of tracing data. In general, the higher the level, the more detailed the information. The following integer levels are available:

Level	Meaning
0	Disables archived redo log tracing (default setting)
1	Tracks archiving of log files
2	Tracks archive status by archive log file destination
4	Tracks archive operational phase
8	Tracks archive log destination activity
16	Tracks detailed archive log destination activity
32	Tracks archive log destination parameter modifications
64	Tracks <code>ARCn</code> process state activity
128	Tracks FAL server process activity
256	Track RFS Logical Client
512	Tracks LGWR redo shipping network activity
1024	Tracks RFS physical client
2048	Tracks RFS/ <code>ARCn</code> ping heartbeat
4096	Tracks real-time apply activity
8192	Tracks Redo Apply activity (media recovery or physical standby)

You can combine tracing levels by setting the value of the `LOG_ARCHIVE_TRACE` parameter to the sum of the individual levels. For example, setting the parameter to 6 generates level 2 and level 4 trace output.

The following are examples of the `ARC0` trace data generated on the primary site by the archiving of log file 387 to two different destinations: the service `standby1` and the local directory `/oracle/dbs`.

Note: The level numbers do not appear in the actual trace output; they are shown here for clarification only.

Level Corresponding entry content (sample)

```

-----
( 1)  ARC0: Begin archiving log# 1 seq# 387 thrd# 1
( 4)  ARC0: VALIDATE
( 4)  ARC0: PREPARE
( 4)  ARC0: INITIALIZE
( 4)  ARC0: SPOOL
( 8)  ARC0: Creating archive destination 2 : 'standby1'
(16)  ARC0: Issuing standby Create archive destination at 'standby1'
( 8)  ARC0: Creating archive destination 1 : '/oracle/dbs/dlarc1_387.log'
(16)  ARC0: Archiving block 1 count 1 to : 'standby1'
(16)  ARC0: Issuing standby Archive of block 1 count 1 to 'standby1'
(16)  ARC0: Archiving block 1 count 1 to : '/oracle/dbs/dlarc1_387.log'
( 8)  ARC0: Closing archive destination 2 : standby1
(16)  ARC0: Issuing standby Close archive destination at 'standby1'
( 8)  ARC0: Closing archive destination 1 : /oracle/dbs/dlarc1_387.log
( 4)  ARC0: FINISH
( 2)  ARC0: Archival success destination 2 : 'standby1'
( 2)  ARC0: Archival success destination 1 : '/oracle/dbs/dlarc1_387.log'
( 4)  ARC0: COMPLETE, all destinations archived
(16)  ARC0: ArchivedLog entry added: /oracle/dbs/dlarc1_387.log
(16)  ARC0: ArchivedLog entry added: standby1
( 4)  ARC0: ARCHIVED
( 1)  ARC0: Completed archiving log# 1 seq# 387 thrd# 1

(32)  Propagating archive 0 destination version 0 to version 2
      Propagating archive 0 state version 0 to version 2
      Propagating archive 1 destination version 0 to version 2
      Propagating archive 1 state version 0 to version 2
      Propagating archive 2 destination version 0 to version 1
      Propagating archive 2 state version 0 to version 1
      Propagating archive 3 destination version 0 to version 1
      Propagating archive 3 state version 0 to version 1
      Propagating archive 4 destination version 0 to version 1
      Propagating archive 4 state version 0 to version 1

(64)  ARCH: changing ARC0 KCRNOARCH->KCRRSCHED
      ARCH: STARTING ARCH PROCESSES
      ARCH: changing ARC0 KCRRSCHED->KCRRSTART
      ARCH: invoking ARC0
      ARC0: changing ARC0 KCRRSTART->KCRRACTIVE
      ARCH: Initializing ARC0
      ARCH: ARC0 invoked
      ARCH: STARTING ARCH PROCESSES COMPLETE
      ARC0 started with pid=8
      ARC0: Archival started

```

The following is the trace data generated by the RFS process on the standby site as it receives archived redo log file 387 in directory /stby and applies it to the standby database:

```

level   trace output (sample)
-----
( 4)    RFS: Startup received from ARCH pid 9272
( 4)    RFS: Notifier
( 4)    RFS: Attaching to standby instance
( 1)    RFS: Begin archive log# 2 seq# 387 thrd# 1
(32)    Propagating archive 5 destination version 0 to version 2
(32)    Propagating archive 5 state version 0 to version 1
( 8)    RFS: Creating archive destination file: /stby/parc1_387.log
(16)    RFS: Archiving block 1 count 11

```

```
( 1)      RFS: Completed archive log# 2 seq# 387 thrd# 1
( 8)      RFS: Closing archive destination file: /stby/parc1_387.log
(16)      RFS: ArchivedLog entry added: /stby/parc1_387.log
( 1)      RFS: Archivelog seq# 387 thrd# 1 available 04/02/99 09:40:53
( 4)      RFS: Detaching from standby instance
( 4)      RFS: Shutdown received from ARCH pid 9272
```

Index

A

activating

- a clone database, 12-31
- a logical standby database, 7-17, 15-4, B-6
- a physical standby database, 7-10, 10-6, 12-31, 12-32, 12-38, 15-4
- a read/write physical standby database, 12-29

activating a clone database, 12-31

adding

- datafiles, 8-6, A-11, A-12
- indexes on logical standby databases, 1-10, 2-3, 9-16
- new or existing standby databases, 1-6
- online redo log files, 5-23, 8-12
- standby redo log, 3-2
- standby redo log group members, 5-23
- tablespaces, 8-6

adjusting

- initialization parameter file
- for logical standby database, 4-5

AFFIRM attribute, 5-19, 14-2

ALTER DATABASE statement

- ABORT LOGICAL STANDBY clause, 15-4
- ACTIVATE STANDBY DATABASE clause, 7-10, 7-17, 10-6, 12-31, 12-38, 15-4
- ACTIVATE STANDBY DATABASE statement, 12-31
- ADD STANDBY LOGFILE clause, 3-3, 8-12, 15-1, A-2
- ADD STANDBY LOGFILE GROUP clause, 3-3
- ADD STANDBY LOGFILE MEMBER clause, 5-23, 15-1, A-2
- ADD SUPPLEMENTAL LOG DATA clause, 15-1
- ALTER STANDBY LOGFILE clause, 3-3
- ALTER STANDBY LOGFILE GROUP clause, 3-3
- CLEAR UNARCHIVED LOGFILES clause, 8-14
- COMMIT TO SWITCHOVER clause, 7-8, 7-11, 7-15, 7-16, 12-39, 15-2
- in Real Application Clusters, D-6
- troubleshooting, A-5, A-6
- CREATE CONTROLFILE clause, 8-14
- CREATE DATAFILE AS clause, A-2
- CREATE STANDBY CONTROLFILE clause, 3-8, A-3
- REUSE clause, 15-2

DROP LOGFILE clause, A-2

DROP STANDBY LOGFILE MEMBER

- clause, 15-2, A-2

FORCE LOGGING clause, 2-5, 3-2, 12-41, 15-2

MOUNT STANDBY DATABASE clause, 15-2

OPEN READ ONLY clause, 15-2

OPEN RESETLOGS clause, 3-8, 8-14

PREPARE TO SWITCHOVER clause, 7-14, 15-2

RECOVER MANAGED STANDBY DATABASE

- clause, 3-11, 4-7, 6-4, 8-5, 12-15, 12-38, 15-3
- background process, 6-4, 8-2
- canceling, 6-5
- canceling log apply services, 8-4
- controlling Redo Apply, 6-4
- failover, 15-4
- foreground session, 6-4
- initiating failover, 7-11
- overriding the delay interval, 6-3
- starting real time apply, 6-5
- switchover scenario, 12-38

REGISTER LOGFILE clause, 15-3, A-4

REGISTER LOGICAL LOGFILE clause, 12-19

RENAME FILE clause, 8-10, A-2

SET STANDBY DATABASE clause

- TO MAXIMIZE AVAILABILITY clause, 15-3
- TO MAXIMIZE PERFORMANCE clause, 7-6
- TO MAXIMIZE PROTECTION clause, 15-3

START LOGICAL STANDBY APPLY clause, 6-5, 7-18, 11-6, A-9

- IMMEDIATE keyword, 6-5, 9-14

- NEW PRIMARY keyword, 7-18

- starting SQL Apply, 4-7

STOP LOGICAL STANDBY APPLY clause, 6-5, 7-17, 11-6, 12-19, 15-4

ALTER SESSION DATABASE DISABLE GUARD statement, 9-16

ALTER SESSION DISABLE GUARD statement

- overriding the database guard, 9-16

ALTER SESSION GUARD DISABLE

- disabling the database guard to define a database link, 7-18

ALTER SESSION statement

- GUARD ENABLE clause, 15-4

ALTER SYSTEM statement

- ARCHIVE LOG CURRENT clause, 12-20, 12-21, 12-22, 12-23, 12-30, 12-40, B-2, B-4, B-6, F-3,

- F-19
 - SWITCH LOGFILE clause, 3-12
- ALTER TABLESPACE statement, 8-11, 12-42, A-12
 - FORCE LOGGING clause, 8-12
- alternate archive destinations
 - setting up initialization parameters for, A-3
- ALTERNATE attribute, 14-4
 - LOG_ARCHIVE_DEST_n initialization parameter, A-3
 - LOG_ARCHIVE_DEST_STATE_n initialization parameter, 5-3
- ANALYZER process, 9-2
- APPLIED_SCN column
 - of V\$LOGSTDBY_PROGRESS view, 12-17
- APPLIER process, 9-2
- applying
 - redo data immediately, 6-2
 - redo data on standby database, 1-3, 1-4, 2-1, 6-1
 - SQL statements to logical standby databases, 6-5
- applying state, 9-12
- AQ_TM_PROCESSES dynamic parameter, A-6
- ARCH attribute, 14-6
 - LOG_ARCHIVE_DEST_n initialization parameter setting for data protection, 5-19
- archive destinations
 - alternate, A-3
- archive gap
 - registering missing log files, 5-28
- archive gaps
 - causes of, 12-43
 - defined, 5-25
 - finding with DBA_LOGSTDBY_LOG view, 5-28
 - finding with V\$ARCHIVE_GAP view, 5-27
 - identifying the logs, 12-45
 - manually applying redo logs to standby database, 12-47
 - manually copying the logs in, 12-46
 - manually resolving on a physical standby database, 5-27
 - preventing, 12-44
 - registering missing log files, 5-28
- ARCHIVE LOG CURRENT clause
 - of ALTER SYSTEM, 12-20, 12-21, 12-22, 12-23, 12-30, 12-40, B-2, B-4, B-6, F-3, F-19
- archive tracing
 - standby databases and, 5-30, G-1
- archived redo log files
 - accessing information about, 8-18, 8-20
 - affected by COMPATIBLE initialization parameter, 5-21
 - applying
 - Redo Apply technology, 1-4
 - SQL Apply technology, 1-4
 - delaying application, 12-37, 14-8
 - on the standby database, 6-3
 - deleting unneeded, 9-12
 - destinations
 - disabling, 5-3
 - displaying with DBA_LOGSTDBY_LOG view, 12-17
 - displaying with V\$ARCHIVE_DEST_STATUS view, 16-1
 - enabling, 5-3
 - determining the most recently archived, 5-29
 - finding gaps on logical standby databases, 5-28
 - finding gaps on physical standby databases, 5-27
 - listing, 12-17
 - managing gaps, 1-10, 5-25
 - See also* gap management
 - manually transferring, 2-5
 - manually transmitting, 12-46
 - overview, 2-8
 - redo data transmitted, 1-4, 6-1
 - registering, 5-28, 12-19
 - during failover, 7-16
 - partial, 12-19
 - retrieving missing, 12-18
 - specifying
 - directory location on the standby database, 5-21
 - standby databases and, 6-5, 6-6, 8-19
 - temporary storage of, 5-2
 - troubleshooting switchover problems, A-4
 - verifying the contents of, 14-28
 - archived redo log repository, 5-2
 - ARCHIVELOG mode
 - software requirements, 2-5
 - archiver process (ARCn)
 - defined, 5-8
 - verifying the contents of completed archived redo log files, 14-28
 - archiving
 - real-time apply, 6-2
 - specifying
 - failure resolution policies for, 5-16, 14-22
 - network transmission modes for, 5-10
 - with the STANDBY_ARCHIVE_DEST initialization parameter, 5-7
 - starting, 3-12
 - to failed destinations, 5-16, 14-22
 - See also* archived redo log files
 - See also* redo transport services
 - ARCn process
 - See* archiver process (ARCn)
 - ASM
 - See* Automatic Storage Management (ASM)
 - ASYNC attribute, 14-23
 - LOG_ARCHIVE_DEST_n initialization parameter setting for data protection, 5-19
 - Asynchronous AutoLog, 5-3
 - asynchronous redo transmission, 5-12, 14-6, 14-23
 - attributes
 - deprecated for the LOG_ARCHIVE_DEST_n initialization parameter, 14-1
 - authentication
 - checks using SYS credentials, 5-13
 - to transfer redo data, 5-13
 - automatic detection of missing log files, 1-3, 1-10, 5-25
 - automatic failover, 1-5, 7-1

Automatic Storage Management (ASM)
 creating a standby database that uses, 12-48
automatic switchover, 1-5, 7-1
 See also switchovers

B

BACKUP INCREMENTAL FROM SCN command

 scenarios using, 12-33

backup operations

 after failovers, 7-12, 7-19

 after unrecoverable operations, 12-43

 configuring on a physical standby database, 1-2
 datafiles, 12-42

 DB_FILE_NAME_CONVERT option of the RMAN
 DUPLICATE command, F-4

 offloading on the standby database, 1-10

 primary databases, 1-2

 used by the broker, 1-6

 using RMAN, 10-1

basic readable standby database *See* simulating a
 standby database environment

batch processing

 on a logical standby database, 9-4

benefits

 Data Guard, 1-9

 logical standby database, 2-3

 of a rolling upgrade, 11-1

 physical standby database, 2-2

 standby redo logs versus archived redo logs, 2-9

BFILE data types

 in logical standby databases, C-2

BINARY_DEGREE data types

 in logical standby databases, C-1

BINARY_FLOAT data types

 in logical standby databases, C-1

BLOB data types

 in logical standby databases, C-1

broker

 command-line interface, 1-10

 defined, 1-5

 graphical user interface, 1-10

BUILDER process, 9-2

C

CANCEL option

 managed recovery and, 8-2

canceling

 log apply services, 8-4

cascaded destinations

 defined, E-1

 for logical standby databases, E-3

 for physical standby databases, E-1, E-2

 materialized view on logical standby
 databases, E-6

 require standby redo log files, 2-9

 role transitions, E-4

 Redo Apply, E-4

 SQL Apply, E-4

 scenario, E-4 to E-7

CHAR data types

 in logical standby databases, C-1

checklist

 tasks for creating physical standby databases, 3-7

 tasks for creating standby databases, 4-3

checkpoints

 V\$LOGSTDBY_PROGRESS view, 9-3

chunking

 transactions, 9-2

CJQ0 process, A-6

CLEAR UNARCHIVED LOGFILES clause

 of ALTER DATABASE, 8-5, 8-14

CLOB data types

 in logical standby databases, C-1

clone database

 ACTIVATE STANDBY DATABASE clause, 12-31

 activating, 12-31

 activating a physical standby database, 12-29

 creating a restore point for, 12-30

 creating restore point for, 12-30

 DB_RECOVERY_FILE_DEST initialization
 parameter, 12-30

 DB_RECOVERY_FILE_DEST_SIZE initialization
 parameter, 12-30

clone databases

 converting a physical standby database to, 12-29

cloning

 a physical standby database, 12-29

 activating a read/write database, 12-31

 CREATE RESTORE POINT command, 12-30

 creating a restore point, 12-30

cluster tables

 in logical standby databases, C-2

collections data types

 in logical standby databases, C-2

command-line interface

 broker, 1-10

commands, Recovery Manager

 DUPLICATE, F-2

COMMIT TO SWITCHOVER clause

 of ALTER DATABASE, 7-8, 7-11, 7-15, 7-16,
 12-39, 15-2

 in Real Application Clusters, D-6

 troubleshooting, A-5, A-6

COMMIT TO SWITCHOVER TO PRIMARY clause

 of ALTER DATABASE, 7-16

communication

 between databases in a Data Guard

 configuration, 1-1

COMPATIBLE initialization parameter

 effect on directory location for archived redo log
 files, 5-21

 setting for a rolling upgrade, 11-2, 11-3, 11-10

complementary technologies, 1-8

configuration options

 creating with Data Guard broker, 1-5

 of redo logs, 2-9

 overview, 1-1

 physical standby databases

- location and directory structure, 2-6
- requirements for password files, 5-13
- standby databases
 - archived redo log repository, 5-2
 - delayed application of archived redo log files on, 12-37
 - delayed standby, 6-3
 - typical, 1-3
- configuring
 - backups on standby databases, 1-2
 - cascaded destinations, E-2
 - disaster recovery, 1-2
 - initialization parameters
 - for alternate archive destinations, A-3
 - for physical standby database, 3-8
 - to create a standby database with a time lag, 12-37
 - to set up redo transport services, 5-4
 - listener for physical standby databases, 3-11
 - no data loss, 1-5
 - physical standby databases, 2-6
 - redo logs, 2-9
 - reporting operations on a logical standby database, 1-2
 - standby databases at remote locations, 1-2
 - standby redo log files, 3-2
 - standby redo log groups, 3-2
- constraints
 - handled on a logical standby database, 9-20
- Context
 - unsupported data types, C-2
- Context data types
 - in logical standby databases, C-2
- control files
 - copying, 3-10
 - creating for standby databases, 3-8
 - modifying with ALTER DATABASE RENAME FILE statement, 8-10
 - planning growth of, 5-23
 - switchover and, 7-8
- CONTROL_FILE_RECORD_KEEP_TIME
 - initialization parameter, 5-24
- converting
 - a logical standby database to a physical standby database
 - aborting, 4-5
 - a physical standby database to a logical standby database, 4-5
- COORDINATOR process, 9-2
 - LSP background process, 9-2
- copying
 - control files, 3-10
- CREATE CONTROLFILE clause
 - of ALTER DATABASE, 8-14
- CREATE DATABASE statement
 - FORCE LOGGING clause, 12-41
- CREATE DATAFILE AS clause
 - of ALTER DATABASE, A-2
- CREATE RESTORE POINT command, 12-30
- CREATE RESTORE POINT statement

- activating a clone database, 12-30
- CREATE STANDBY CONTROLFILE clause
 - of ALTER DATABASE, 3-8, 15-2, A-3
- CREATE TABLE AS SELECT (CTAS) statements
 - applied on a logical standby database, 9-4
- creating
 - a new password file, 4-5
 - database link, 7-18
 - indexes on logical standby databases, 9-16
 - standby database
 - that use OMF, F-7, F-8
 - standby redo log files, 3-3
 - traditional initialization parameter file
 - for physical standby database, 3-8
- cross-instance archival
 - in Real Application Clusters configurations, D-3
 - standby redo log files and, D-3
 - using the log writer process, D-3

D

- data availability
 - balancing against system performance requirements, 1-10
- data corruption
 - safeguarding against, 1-9
- Data Guard broker
 - defined, 1-5
 - distributed management framework, 7-1
 - failovers, 1-6
 - fast-start, 7-1
 - manual, 1-6, 7-1
 - fast-start failover, 1-6
 - switchovers, 7-1
- Data Guard configurations
 - archiving to standby destinations using the archive process, 5-4
 - archiving to standby destinations using the log writer process, 5-11, 6-2
 - defined, 1-1
 - protection modes, 1-7
 - redo transport services and, 5-1
 - upgrading Oracle Database software, B-1
- data loss
 - due to failover, 1-5
 - minimizing, 7-10
 - switchover and, 7-2
- data protection
 - balancing against performance, 1-10
 - benefits, 1-9
 - ensuring no data loss, 2-2
 - flexibility, 1-10
 - provided by Data Guard, 1-1
- data protection modes
 - affect on network timeouts, 14-19
 - enforced by redo transport services, 1-3
 - influence on network reconnection, 14-19
 - maximum availability mode, 5-19
 - maximum performance mode, 5-19
 - maximum protection mode, 5-19

- minimum set of requirements, 5-19
 - overview, 1-7
 - setting up synchronous and asynchronous network I/O operations, 14-23
 - specifying, 5-19
- Data Pump utility
 - using transportable tablespaces with physical standby databases, 8-10
- data types
 - BFILE, C-2
 - BINARY_DEGREE, C-1
 - BINARY_FLOAT, C-1
 - BLOB, C-1
 - CHAR, C-1
 - CLOB, C-1
 - collections in logical standby databases, C-2
 - DATE, C-1
 - encrypted columns, C-2
 - INTERVAL, C-1
 - LONG, C-1
 - LONG RAW, C-1
 - NCHAR, C-1
 - NCLOB, C-1
 - NUMBER, C-1
 - NVARCHAR2, C-1
 - RAW, C-1
 - ROWID, C-2
 - Spatial, Image, and Context, C-2
 - TIMESTAMP, C-2
 - UROWID, C-2
 - user-defined, C-2
 - VARCHAR, C-2
 - VARCHAR2, C-2
 - XMLType, C-2
- database guard, 6-2, 9-16
 - overriding, 9-16
- database incarnation
 - changes with OPEN RESETLOGS, 8-13
- database link
 - creating, 7-18
- database roles
 - primary, 1-2, 7-1
 - standby, 1-2, 7-1
 - transitions, 1-5
- database schema
 - physical standby databases, 1-2
- Database Upgrade Assistant (DBUA), B-1
- databases
 - cascading standby databases *See* cascaded destinations
 - cloning, 12-29
 - failover and, 7-6
 - primary *See* primary database
 - role transition and, 7-1
 - surviving disasters and data corruptions, 1-1
 - upgrading software versions, 11-1
 - using password files, 5-13
- datafiles
 - adding to primary database, 8-6
 - deleting from the primary database, 8-9
 - monitoring, 8-14, 12-41
 - renaming on the primary database, 8-11
- DATE data types
 - in logical standby databases, C-1
- DB_FILE_NAME_CONVERT initialization parameter
 - location for transportable tablespaces, 8-10
- DB_FILE_NAME_CONVERT option
 - RMAN DUPLICATE command, F-4
- DB_NAME initialization parameter, 3-5
- DB_RECOVERY_FILE_DEST initialization parameter
 - setting for clone database, 12-30
 - setting up recovery areas, 5-5
- DB_RECOVERY_FILE_DEST_SIZE initialization parameter
 - setting for clone database, 12-30
- DB_ROLE_CHANGE system event, 7-5, 7-7
- DB_UNIQUE_NAME attribute, 14-7
- DB_UNIQUE_NAME initialization parameter, A-7
 - required for shared flash recovery areas, 5-7
 - required with LOG_ARCHIVE_CONFIG parameter, 13-2
 - setting database initialization parameters, 3-4
- DBA_DATA_FILES view, 8-14
- DBA_LOGMNR_PURGED_LOG view
 - list archived redo log files that can be deleted, 9-12
- DBA_LOGSTDBY_EVENTS view, 9-5, 16-1, A-9
 - capturing logical standby, 11-3
- DBA_LOGSTDBY_HISTORY view, 16-1
- DBA_LOGSTDBY_LOG view, 9-5, 16-1
 - finding gaps on logical standby databases, 5-28
 - listing archived redo log files, 12-17
- DBA_LOGSTDBY_NOT_UNIQUE view, 16-1
- DBA_LOGSTDBY_PARAMETERS view, 16-1
- DBA_LOGSTDBY_SKIP view, 16-1
 - determining SQL Apply support for schemas, C-3
- DBA_LOGSTDBY_SKIP_TRANSACTION view, 16-1
- DBA_LOGSTDBY_UNSUPPORTED view, 16-1, C-3
- DBA_TABLESPACES view, 8-14
- DBMS_ALERT, C-3
- DBMS_AQ, C-3
- DBMS_DESCRIBE, C-3
- DBMS_JAVA, C-3
- DBMS_JOB, C-3
- DBMS_LOB, C-3
- DBMS_LOGSTDBY package
 - INSTANTIATE_TABLE procedure, 9-18
 - SKIP procedure, A-9
 - SKIP_ERROR procedure, A-4
 - SKIP_TRANSACTION procedure, A-9
- DBMS_LOGSTDBY procedure
 - capturing events in DBA_LOGSTDBY_EVENTS table, 11-3
- DBMS_LOGSTDBY.BUILD procedure
 - building a dictionary in the redo data, 4-4
- DBMS_LOGSTDBY.BUILD subprogram
 - uses Flashback Query, 4-4
- DBMS_METADATA, C-3

- DBMS_OBFUSCATION_TOOLKIT, C-3
- DBMS_OUTPUT, C-3
- DBMS_PIPE, C-3
- DBMS_RANDOM, C-3
- DBMS_REDEFINITION, C-3
- DBMS_REFRESH, C-3
- DBMS_REGISTRY, C-3
- DBMS_SCHEDULER, C-3
- DBMS_SPACE_ADMIN, C-3
- DBMS_SQL, C-3
- DBMS_TRACE, C-3
- DBMS_TRANSACTION, C-3
- DBSNMP process, A-6
- DDL statements
 - supported by SQL Apply, C-1
- DDL transactions
 - applied on a logical standby database, 9-4
 - applying to a logical standby database, 9-4
- DEFER attribute
 - LOG_ARCHIVE_DEST_STATE_n initialization parameter, 5-3, 12-40
- DELAY attribute, 14-8
 - LOG_ARCHIVE_DEST_n initialization parameter, 6-3, 12-37
- DELAY option
 - of ALTER DATABASE RECOVER MANAGED STANDBY DATABASE
 - cancelling, 6-3
- delaying
 - application of archived redo log files, 12-37, 14-8
 - application of redo log files, 6-3
- deleting
 - archived redo log files
 - indicated by the DBA_LOGMNR_PURGED_LOG view, 9-12
 - not needed by SQL Apply, 9-12
 - datafiles, 8-9
 - online redo log files, 8-12
- DEPENDENCY attribute, 14-10
 - LOG_ARCHIVE_DEST_n initialization parameter, 5-24
- deprecated attributes
 - on the LOG_ARCHIVE_DEST_n initialization parameter, 14-1
- destinations
 - archived redo log repository, 5-2
 - cross-instance archival, D-3
 - displaying with V\$ARCHIVE_DEST view, 16-1
 - Oracle Change Data Capture archival, 5-3
 - Oracle Streams archival, 5-2
 - role-based definitions, 14-26
 - shared, 5-24, 14-10
 - specifying, 5-3
 - verifying settings for role transitions, 12-9
- detecting
 - missing archived redo log files, 1-3, 1-10, 5-25
 - network disconnects between primary and standby databases, 14-19
- determining
 - the highest applicable (newest) SCN, 12-17
- DG_CONFIG attribute, 14-7
 - on the LOG_ARCHIVE_CONFIG initialization parameter, 5-20
- DGMGRL command-line interface
 - invoking failovers, 1-6, 7-1
 - simplifying switchovers, 1-6, 7-1
- dictionary
 - building a LogMiner, 4-4
- direct path inserts
 - SQL Apply DML considerations, 9-4
- directory locations
 - archived redo log files, 5-21
 - Optimal Flexible Architecture (OFA), 2-6
 - set up with ASM, 2-6
 - set up with OMF, 2-6
 - specifying with STANDBY_ARCHIVE_DEST initialization parameter, 5-21
 - structure on standby databases, 2-6
- disabling
 - a destination for archived redo log files, 5-3
 - database guard to define a database link, 7-18
- disaster recovery
 - benefits, 1-9
 - configuring, 1-2
 - provided by Data Guard, 1-1
 - provided by standby databases, 1-2
 - ReadMe file at standby site, 12-11
- DISCONNECT FROM SESSION, 8-2
- disk I/O
 - controlling with the AFFIRM and NOAFFIRM attributes, 14-2
- DML
 - batch updates on a logical standby database, 9-4
- DML transactions
 - applying to a logical standby database, 9-4
- downstream capture database
 - Oracle Streams and Data Guard redo transport services, 5-3
- DROP STANDBY LOGFILE clause
 - of ALTER DATABASE, A-2
- DROP STANDBY LOGFILE MEMBER clause
 - of ALTER DATABASE, 15-2, A-2
- dropping
 - datafiles
 - examples, 8-9
 - online redo log files, 8-12
 - tablespaces from primary database, 8-9
- dynamic parameters
 - AQ_TM_PROCESSES, A-6
 - JOB_QUEUE_PROCESSES, A-6
- dynamic performance views, 8-15
 - See also* views

E

- ENABLE attribute
 - LOG_ARCHIVE_DEST_STATE_n initialization parameter, 5-3, 12-40
- enabling
 - database guard on logical standby

- databases, 15-4
- destinations for archived redo log files, 5-3
- real-time apply, 8-5
 - on logical standby databases, 6-5, 9-14
 - on physical standby databases, 6-5
- encrypted columns
 - logical standby databases, C-2
- extensible indexes
 - supported by logical standby databases, C-2

F

- failovers, 1-5
 - and cascaded configurations, E-4
 - choosing a target standby database, 12-9, 12-16
 - Data Guard broker, 1-6, 7-1
 - defined, 1-5, 7-2
 - displaying history with DBA_LOGSTDBY_HISTORY, 16-1
 - fast-start failover, 7-1
 - FINISH FORCE, 7-11
 - flashing back databases after, 7-20
 - logical standby databases and, 7-16, 12-16
 - manual versus automatic, 1-5, 7-1
 - minimal data loss and, 12-18
 - minimal performance impact, 12-18
 - performing backups after, 7-12, 7-19
 - physical standby databases and, 7-9, 15-4
 - preparing for, 7-6
 - re-creating after, 7-10
 - simplifying with Data Guard broker, 7-1
 - transferring redo data before, 7-6
 - viewing characteristics for logical standby databases, 9-6
 - with maximum performance mode, 7-6
 - with maximum protection mode, 7-6
- failure resolution policies
 - specifying for redo transport services, 5-16, 14-22
- FAL_CLIENT initialization parameter, 5-27
- FAL_SERVER initialization parameter, 5-27
- false network failure detection, 14-19, 14-20
- fast-start failover
 - automatic failover, 1-6, 7-1
 - enhancements in Oracle Enterprise Manager, xx
 - monitoring, 8-14
- finding
 - missing log files, 5-27
- fixed views
 - See* views
- flash recovery areas
 - default location, 5-5, 5-6, 14-12
 - setting up, 5-5
 - sharing among multiple databases, 5-7
 - STANDBY_ARCHIVE_DEST=LOCATION parameter, 5-7
- Flashback Database
 - after a role transition, 7-20
 - after OPEN RESETLOGS, 12-27
 - after role transitions, 7-20
 - characteristics complementary to Data Guard, 1-9

- clone database, 12-29
- enabling for guaranteed restore point, 12-30
- logical standby database, 12-25
- physical standby database, 12-24
- Flashback Query
 - used by DBMS_LOGSTDBY.BUILD subprogram, 4-4
- FORCE keyword
 - on RECOVER MANAGED STANDBY DATABASE FINISH, 7-11
- FORCE LOGGING clause
 - of ALTER DATABASE, 2-5, 3-2, 12-41, 15-2
 - of ALTER TABLESPACE, 8-12
 - of CREATE DATABASE, 12-41

G

- gap management, 5-25
 - automatic detection and resolution, 1-3, 1-10
 - defined, 5-25
 - detecting missing log files, 1-10
 - registering archived redo log files, 5-28
 - during failover, 7-16
 - See also* archived redo log files
- gap sequences
 - determining, 5-27, 5-28
- global dynamic performance views, 8-15
 - See also* views
- guaranteed restore point
 - creating, 12-30
 - enable the flashback recovery area for Flashback Database, 12-30
 - setting DB_RECOVERY_FILE_DEST initialization parameter, 12-30
 - setting DB_RECOVERY_FILE_DEST_SIZE initialization parameter, 12-30
- GUARD DISABLE clause
 - of ALTER SESSION, 7-18
- GUARD ENABLE clause
 - of ALTER SESSION, 15-4
- GV\$ fixed views, 8-15
 - See also* views
- GV\$INSTANCE view, D-6

H

- hardware
 - requirements for Data Guard configurations, 2-4
- heap-organized tables
 - in logical standby databases, C-2
- high availability
 - benefits, 1-9
 - provided by Data Guard, 1-1
 - provided by RAC and Data Guard, 1-8

I

- idle state, 9-12
- Image data types
 - in logical standby databases, C-2
- incarnation of a database

- changed, 8-13
- index-organized tables
 - in logical standby databases, C-2
- initialiation parameters
 - DB_UNIQUE_NAME, 3-4
- initialization parameter file
 - creating from server parameter file
 - for physical standby database, 3-8
 - modifying
 - for physical standby database, 3-8
- initialization parameters
 - CONTROL_FILE_RECORD_KEEP_TIME, 5-24
 - DB_FILE_NAME_CONVERT, F-4
 - DB_UNIQUE_NAME, A-7
 - FAL_CLIENT, 5-27
 - FAL_SERVER, 5-27
 - LOG_ARCHIVE_DEST, 12-47
 - LOG_ARCHIVE_DEST_STATE_n, 5-3
 - LOG_ARCHIVE_FORMAT, 5-22
 - LOG_ARCHIVE_MIN_SUCCEED_DEST, 14-14
 - LOG_ARCHIVE_TRACE, 5-30, G-1, G-2
 - LOG_FILE_NAME_CONVERT, F-5
 - modifying for physical standby databases, 3-8
 - setting for both the primary and standby roles, 14-26
 - STANDBY_ARCHIVE_DEST, 5-21
 - USER_DUMP_DEST, G-1
- INITIALIZING state, 9-11
- INSTANTIATE_TABLE procedure
 - of DBMS_LOGSTDBY, 9-18
- INTERVAL data types
 - in logical standby databases, C-1

J

JOB_QUEUE_PROCESSES dynamic parameter, A-6

L

- latency
 - on logical standby databases, 9-4
- LGWR attribute, 14-6
 - asynchronous redo transmission, 5-12, 14-6, 14-23
 - LOG_ARCHIVE_DEST_n initialization parameter
 - setting for data protection, 5-19
- LGWR process
 - See log writer process (LGWR)
- listener.ora file
 - configuring, 3-11
 - redo transport services tuning and, A-9
 - troubleshooting, 12-40, A-2, A-9
- listing
 - archived redo log files, 12-17
- loading dictionary state, 9-11
- LOCATION attribute, 14-12
 - setting
 - flash recovery area with USE_DB_RECOVERY_FILE_DEST, 5-6
 - LOG_ARCHIVE_DEST_n initialization parameter, A-3

- log apply services
 - defined, 1-4, 6-1
 - delaying application of redo data, 6-3, 12-37, 14-8
 - real-time apply
 - and standby redo log files, 2-8
 - defined, 6-1, 6-2
 - monitoring with LOG_ARCHIVE_TRACE, G-2
 - monitoring with V\$ARCHIVE_DEST_STATUS view, 8-20
- Redo Apply
 - defined, 6-1, 6-4
 - monitoring, 6-5, 8-15, 8-19
 - starting, 6-4, 8-1
 - stopping, 6-5, 8-2
 - tuning the log apply rate, 8-22
- SQL Apply
 - defined, 1-4, 6-1, 6-2
 - monitoring, 6-6
 - starting, 6-5
 - stopping, 6-5
 - tuning for Redo Apply, 8-22
- log writer process (LGWR)
 - ASYNC network transmission, 5-10, 14-23
 - defined, 5-10
 - local archiving, 5-4
 - NET_TIMEOUT attribute, 14-19
 - reconnecting after a network timeout, 14-19
 - SYNC network transmission, 14-23
- LOG_ARCHIVE_CONFIG initialization parameter, 3-4, 3-5, 3-8, 5-15, 13-2
 - example, 14-7
 - listing unique database names defined with, 16-2
 - relationship to DB_UNIQUE_NAME parameter, 13-2
 - relationship to DG_CONFIG attribute, 14-7
 - with the DG_CONFIG attribute, 5-20
- LOG_ARCHIVE_DEST_10 initialization parameter
 - default flash recovery area, 5-5, 5-6, 14-12
- LOG_ARCHIVE_DEST_n initialization parameter, 5-10
 - AFFIRM attribute, 5-19, 14-2
 - ALTERNATE attribute, 14-4, A-3
 - ARCH attribute, 5-19, 14-6
 - ASYNC attribute, 14-23
 - setting for data protection, 5-19
 - DB_UNIQUE_NAME attribute, 14-7
 - DELAY attribute, 6-3, 12-37, 14-8
 - DEPENDENCY attribute, 5-24, 14-10
 - deprecated attributes, 14-1
 - LGWR attribute, 14-6
 - setting for data protection, 5-19
 - LOCATION attribute, 14-12, A-3
 - MANDATORY attribute, 14-14
 - MAX_CONNECTIONS attribute, 14-16
 - MAX_FAILURE attribute, 14-17
 - NET_TIMEOUT attribute, 14-19
 - NOAFFIRM attribute, 14-2
 - NOALTERNATE attribute, A-3
 - NODELAY attribute, 6-3

- NOREGISTER attribute, 14-21
- NOREOPEN attribute, 5-16
- OPTIONAL attribute, 14-14
- overview, 5-3
- QUOTA_SIZE attribute, D-5
- REOPEN attribute, 5-16, 14-22
- SERVICE attribute, 14-12
- setting up recovery areas, 5-5
- SYNC attribute, 14-23
 - setting for data protection, 5-19
- TEMPLATE attribute, 14-24
- VALID_FOR attribute, 5-14, 14-26
- VERIFY attribute, 14-28
- LOG_ARCHIVE_DEST_STATE_n initialization
 - parameter, 5-3
 - ALTERNATE attribute, 5-3
 - DEFER attribute, 5-3, 12-40
 - ENABLE attribute, 5-3, 12-40
 - RESET attribute, 5-3
- LOG_ARCHIVE_FORMAT initialization
 - parameter, 5-22
- LOG_ARCHIVE_MIN_SUCCEED_DEST initialization
 - parameter, 14-14
- LOG_ARCHIVE_TRACE initialization
 - parameter, 5-30, G-1, G-2
- logical change records (LCR)
 - converted by PREPARER process, 9-2
 - exhausted cache memory, 9-3
 - staged, 9-2
- logical corruptions
 - resolving, 1-10
- logical standby databases, 1-2
 - adding
 - datafiles, A-11
 - indexes, 1-10, 2-3, 9-16
 - tables, 9-18
 - archive gaps
 - finding with DBA_LOGSTDBY_LOG view, 5-28
 - manually resolving, 5-28
 - background processes, 9-2
 - benefits, 1-10, 2-3
 - cascading, E-1, E-3
 - creating, 4-1
 - converting from a physical standby database, 4-5
 - with Data Guard broker, 1-5
 - data types
 - supported, C-1
 - unsupported, C-2
 - database guard
 - overriding, 9-16
 - executing SQL statements on, 1-2
 - failovers, 7-16
 - displaying history of, 16-1
 - flashing back after, 12-25
 - handling failures, A-4
 - target of, 12-16
 - viewing characteristics with V\$LOGSTDBY_STATS, 9-6
 - logical standby process (LSP) and, 9-2
 - materialized views
 - creating on, 1-10, 2-3, E-3, E-6
 - support for, C-4
 - monitoring, 6-6, 16-1
 - password file, 4-5
 - read-only operations, 1-10
 - setting VALID_FOR attribute, 12-3
 - SQL Apply, 1-4
 - resynchronizing with primary database branch of redo, 9-20
 - skipping DDL statements, C-4
 - skipping SQL statements, C-4
 - starting real-time apply, 6-5, 9-14
 - stopping, 6-5
 - technology, 6-1
 - transaction size considerations, 9-2
 - starting
 - real-time apply, 6-5
 - states
 - applying, 9-12
 - idle, 9-12
 - initializing, 9-11
 - loading dictionary, 9-11
 - waiting on gaps, 9-12
 - switchovers, 7-13
 - throughput and latency, 9-4
 - UNDO_RETENTION initialization
 - parameter, 4-4
 - upgrading, B-4
 - rolling upgrades, 2-5
- logical standby process (LSP)
 - ARCn archival processing, 5-9
 - COORDINATOR process, 9-2
 - LGWR SYNC archival processing, 5-11
- LogMiner dictionary
 - using DBMS_LOGSTDBY.BUILD procedure to build, 4-4
 - when creating a logical standby database, 4-5
- LONG data types
 - in logical standby databases, C-1
- LONG RAW data types
 - in logical standby databases, C-1

M

- managed recovery operations
 - See* Redo Apply
- managed recovery process (MRP)
 - ARCn archival processing, 5-9
 - launching parallel recovery processes, 5-9
 - LGWR SYNC archival processing, 5-11
 - See also* Redo Apply
- MANDATORY attribute, 14-14
 - LOG_ARCHIVE_DEST_n initialization parameter, 5-22
- materialized views
 - creating on logical standby databases, 1-10, 2-3, E-6
 - on cascaded destinations, E-3

MAX_CONNECTIONS attribute, 14-16
 MAX_FAILURE attribute, 14-17
 maximum availability mode
 defined, 5-18
 influence on network reconnection, 14-19
 introduction, 1-7
 requires standby redo log files, 2-9
 setting, 5-19
 maximum performance mode, 7-6
 introduction, 1-8, 5-19
 setting, 5-19
 maximum protection mode
 for Real Application Clusters, D-5
 influence on network reconnection, 14-19
 introduction, 1-7, 5-18
 requires standby redo log files, 2-9
 setting, 5-19
 standby databases and, 7-6
 media recovery
 parallel, 8-22
 memory
 exhausted LCR cache, 9-3
 missing log sequence
 See also gap management
 detecting, 1-10
 modifying
 a logical standby database, 9-16
 initialization parameters for physical standby
 databases, 3-8
 standby control file, 8-10
 monitoring
 log apply services, 8-20
 primary database events, 8-14
 redo transport services, 5-29
 standby databases, 8-5
 tablespace status, 8-14
 MOUNT STANDBY DATABASE clause
 of ALTER DATABASE, 15-2
 MRP
 See managed recovery process
 multimedia data types
 in logical standby databases, C-2
 unsupported by logical standby databases, C-2

N

NCHAR data types
 in logical standby databases, C-1
 NCLOB data types
 in logical standby databases, C-1
 NET_TIMEOUT attribute, 14-19
 ignored for LGWR ASYNC destinations, 5-13
 network I/O operations
 coordinating timeout parameter values, 14-20
 detecting a disconnect, 14-19
 false failures, 14-19, 14-20
 influence of data protection modes, 14-19
 network timers
 NET_TIMEOUT attribute, 14-19
 setting up synchronous or asynchronous, 14-23

 tuning
 redo transport services, A-9
 network timeouts
 acknowledging, 14-19
 for LGWR ASYNC destinations, 5-13
 NEWEST_SCN column
 of V\$LOGSTDBY_PROGRESS view, 12-17
 no data loss
 benefits, 1-9
 data protection modes overview, 1-7
 ensuring, 1-5, 2-2
 environments, 5-10
 guaranteeing, 1-5
 provided by maximum availability mode, 1-7,
 5-18
 provided by maximum protection mode, 1-7,
 5-18
 NOAFFIRM attribute, 14-2
 NOALTERNATE attribute
 LOG_ARCHIVE_DEST_n initialization
 parameter, A-3
 NODELAY attribute
 LOG_ARCHIVE_DEST_n initialization
 parameter, 6-3
 NODELAY option
 of Redo Apply operations, 12-39
 NOREGISTER attribute, 14-21
 NOREOPEN attribute
 LOG_ARCHIVE_DEST_n initialization
 parameter, 5-16
 NUMBER data types
 in logical standby databases, C-1
 NVARCHAR2 data types
 in logical standby databases, C-1

O

off-site archiving of redo data, 5-2
 OMF
 See Oracle Managed Files (OMF)
 on-disk database structures
 physical standby databases, 1-2
 online redo log files
 adding, 8-12
 archive gap management, 5-25
 deleting, 8-12
 dropping, 8-12
 OPEN READ ONLY clause
 of ALTER DATABASE, 15-2
 OPEN RESETLOGS
 flashing back after, 12-27
 OPEN RESETLOGS clause
 database incarnation change, 8-13
 of ALTER DATABASE, 3-8, 8-14
 recovery, 8-13
 operating systems
 requirements for Data Guard configurations, 2-4
 operational requirements, 2-4, 2-5
 standby databases
 operating system requirements, 2-4

- Optimal Flexible Architecture (OFA)
 - directory structure, 2-6
- OPTIONAL attribute, 14-14
 - LOG_ARCHIVE_DEST_n initialization parameter, 5-22
- ORA-01102 message
 - causing switchover failures, A-6
- Oracle Advanced Security
 - providing secure redo transmission, 5-13, 5-14
- Oracle Automatic Storage Management (ASM), 2-6
- Oracle Change Data Capture archival, 5-3
- Oracle Database
 - requirements for upgrading with SQL Apply, 11-2
 - upgrading, B-1
 - upgrading with SQL Apply, 11-1
- Oracle Database Enterprise Edition
 - software requirements, 2-5
- Oracle databases
 - upgrading, 2-5
- Oracle Enterprise Manager
 - enhancements, xx
 - invoking failovers, 1-6, 7-1
 - invoking switchovers, 1-6, 7-1
- Oracle Managed Files (OMF), 2-6
 - creating a standby database that uses, 12-48
 - creating standby databases that use, F-7, F-8
- Oracle Net
 - communication between databases in a Data Guard configuration, 1-1
- Oracle Recovery Manager utility (RMAN)
 - backing up files on a physical standby database, 10-1
- Oracle Standard Edition
 - simulating a standby database environment, 2-5
- Oracle Streams
 - archival, 5-2
 - downstream capture database, 5-3

P

- pageout considerations, 9-3
- pageouts
 - SQL Apply, 9-3
- Parallel DDL (PDDL) transactions
 - SQL Apply, 9-4
- parallel DML (PDML) transactions
 - SQL Apply, 9-4
- parallel execution processes
 - logical standby databases, 5-9
 - physical standby databases, 5-9
 - tuning the log apply rate, 8-22
 - specifying
 - PARALLEL_MAX_SERVERS initialization parameter, 8-22, 13-3
- PARALLEL option
 - tuning the log apply rate for Redo Apply, 8-22
- parallel recovery processes
 - initiated for Redo Apply, 5-9
- PARALLEL_MAX_SERVERS initialization parameter
 - specifying
 - creating a logical standby database, 8-22, 13-3
- partial archived redo log files
 - registering, 12-19
- password files
 - creating new, 4-5
 - requirements, 5-13
 - setting REMOTE_LOGIN_PASSWORDFILE initialization parameter, 5-14
- patch set releases
 - upgrading, 2-5
- performance
 - balancing against data availability, 1-10
 - balancing against data protection, 1-10
 - monitoring redo transport services, 5-30
- physical standby databases
 - activating as a clone database, 12-31
 - altering
 - online redo log files, 8-12
 - applying redo data, 6-1, 6-4
 - cascading, E-1
 - Redo Apply technology, 6-4
 - applying redo log files
 - starting, 6-4
 - benefits, 2-2
 - choosing a target for failover, 12-10
 - cloning
 - creating a restore point, 12-30
 - for testing, development, and reporting, 12-29
 - configuration options, 2-6
 - converting to a logical standby database, 4-5
 - converting to a read/write database, 12-29
 - creating
 - checklist of tasks, 3-7
 - configuring a listener, 3-11
 - directory structure, 2-7
 - initialization parameters for, 3-8
 - traditional initialization parameter file, 3-8
 - with Data Guard broker, 1-5
 - defined, 1-2
 - failover
 - checking for updates, 7-7
 - finding gaps with V\$ARCHIVE_GAP view, 5-27
 - flashing back after failover, 12-24
 - manually resolving archive gaps, 5-27
 - monitoring, 6-5, 8-5, 8-19, 16-1
 - online backups and, 2-2
 - opening for read-only or read/write access, 8-2
 - read-only, 2-2, 8-2
 - read/write testing and reporting, 12-29
 - recovering through OPEN RESETLOGS, 8-13
 - Redo Apply, 1-4, 2-1
 - resynchronizing with primary database branch of redo, 8-13
 - role transition and, 7-7
 - rolling forward
 - when lagging far behind the primary database, 12-33
 - when nologging changes applied to a small subset of datafiles, 12-34

- when nologging changes are
 - widespread, 12-36
- rolling forward with BACKUP INCREMENTAL FROM SCN command, 12-33
- setting VALID_FOR attribute, 12-1
- shutting down, 8-2
- starting
 - log apply services, 6-4
 - real-time apply, 6-5
- support for DDL, 2-2
- support for DML, 2-2
- synchronizing with the primary database, 12-33
- tuning the log apply rate, 8-22
- upgrading, B-2
- using transportable tablespaces, 8-10
- PL/SQL supplied packages
 - supported, C-3
 - unsupported, C-3
- PREPARE TO SWITCHOVER clause
 - of ALTER DATABASE, 7-14, 15-2
- PREPARER process, 9-2
 - staging LCRs in SGA, 9-2
- primary database
 - backups and, 7-12, 7-19
 - configuring
 - for cross-instance archival, D-3
 - on Real Application Clusters, 1-2
 - single-instance, 1-2
 - datafiles
 - adding, 8-6
 - renaming, 8-9
 - defined, 1-2
 - failover and, 7-2
 - gap resolution, 1-10
 - gathering redo log archival information, 5-29
 - initialization parameters
 - and physical standby database, 3-8
 - monitoring events on, 8-14
 - network connections
 - avoiding network hangs, 14-19
 - detecting disconnections, 14-19
 - handling network timeouts, 14-19
 - preparing for
 - physical standby database creation, 3-1
 - prerequisite conditions for
 - logical standby database creation, 4-1
 - Real Application Clusters and
 - setting up, D-2, D-4
 - redo transport services on, 1-3
 - reducing workload on, 1-10
 - setting archive tracing, 5-30
 - sharing a flash recovery area, 5-7
 - switchover, 7-4
 - switchovers
 - initiating, 7-8
 - tablespaces
 - adding, 8-6
 - dropping, 8-9
- primary databases
 - ARCHIVELOG mode, 2-5

- software requirements, 2-5
- using V\$ARCHIVED_LOG view to find missing log files, 5-27
- primary key columns
 - logged with supplemental logging, 4-4, 9-4
- primary role, 1-2
- processes
 - archiver (ARCn), 5-8
 - CJQ0, A-6
 - DBSNMP, A-6
 - log writer (LGWR), 5-10
 - preventing switchover, A-6
 - QMN0, A-6
 - See also* managed recovery process (MRP)
 - SQL Apply architecture, 9-1, 9-10
- production database
 - See* primary database
- protection modes
 - maximum availability mode, 1-7, 5-18
 - maximum performance mode, 1-8, 5-19
 - maximum protection mode, 1-7, 5-18
 - monitoring, 8-14
 - See* data protection modes

Q

- QMN0 process, A-6
- queries
 - improved performance, 1-10
 - offloading on the standby database, 1-10
- QUOTA_SIZE attribute
 - LOG_ARCHIVE_DEST_n initialization parameter, D-5

R

- RAW data types
 - in logical standby databases, C-1
- raw devices
 - standby redo log files residing on, 2-9
- READER process, 9-2
- read-only operations, 1-4
 - definition, 2-2
 - logical standby databases, 1-10
 - physical standby databases and, 8-2
- read/write databases, 12-29
- Real Application Clusters, D-2
 - characteristics complementary to Data Guard, 1-8
 - cross-instance archival, D-3
 - performing switchover and, D-5, D-6
 - primary databases and, 1-2, D-2, D-4
 - setting
 - maximum data protection, D-5
 - standby databases and, 1-2, D-1, D-3
 - using standby redo log files, 3-3, D-3
 - using standby redo logs, 2-9
- real-time apply
 - and SQL Apply, 9-14
 - defined, 6-1, 6-2
 - monitoring progress in V\$ARCHIVE_DEST_

- STATUS, 8-20
- overview of log apply services, 1-3
- require standby redo log files, 2-9
- RFS process with, 6-1
- starting, 6-5
 - on logical standby, 6-5
- starting on logical standby databases, 6-5, 9-14
- starting on physical standby databases, 6-5
- stopping
 - on logical standby, 6-5
 - on physical standby databases, 8-2
- tracing data with LOG_ARCHIVE_TRACE
 - initialization parameter, G-2
 - using standby redo log files, 2-8
- recommendations for secure redo transmission, 5-13
- reconnecting
 - after a network timeout, 14-19
 - network connection
 - when in maximum availability mode, 14-19
 - when in maximum protection mode, 14-19
- RECOVER MANAGED STANDBY DATABASE
 - CANCEL clause
 - aborting, 4-5
- RECOVER MANAGED STANDBY DATABASE
 - clause
 - canceling the DELAY control option, 6-3
 - FORCE keyword, 7-11
 - of ALTER DATABASE, 3-11, 4-7, 6-4, 8-4, 8-5, 12-15, 12-38, 15-3, 15-4
 - background process, 6-4, 8-2
 - controlling Redo Apply, 6-4
 - foreground session, 6-4
 - initiating failover, 7-11
 - overriding the delay interval, 6-3
 - starting real time apply, 6-5
 - switchover scenario, 12-38
- RECOVER TO LOGICAL STANDBY clause
 - converting a physical standby database to a logical standby database, 4-5
- recovering
 - after a NOLOGGING clause is specified, 12-41
 - from errors, A-10
 - logical standby databases, 9-20
 - physical standby databases
 - after an OPEN RESETLOGS, 8-13
 - through resetlogs, 8-13, 9-20
- Recovery Manager
 - characteristics complementary to Data Guard, 1-9
 - commands
 - DUPLICATE, F-2
 - DB_FILE_NAME_CONVERT option of the
 - DUPLICATE command, F-4
 - standby database
 - creating, F-2, F-5
 - creating standby control files, F-2
 - creating using image copies, F-15
 - DB_FILE_NAME_CONVERT initialization parameter, F-4
 - LOG_FILE_NAME_CONVERT initialization parameter, F-5
 - naming standby datafiles, F-3
 - preparing using RMAN, F-1
 - setting up with OMF, F-7, F-8
 - starting RMAN and standby instance, F-7
- re-creating
 - a table on a logical standby database, 9-18
- Redo Apply
 - defined, 1-4, 2-1, 6-1
 - flashing back after failover, 12-24, 12-25
 - monitoring, 8-20
 - options
 - NODELAY, 12-39
 - parallel recovery processes and MRP, 5-9
 - read/write testing and reporting, 12-29
 - resolving archive gaps, 5-27
 - role transitions and cascaded configurations, E-4
 - starting, 3-11, 6-5
 - stopping, 8-2
 - technology, 1-4
 - tuning the log apply rate, 8-22
- redo data
 - applying
 - through Redo Apply technology, 1-4
 - through SQL Apply technology, 1-4
 - to standby database, 6-1
 - to standby databases, 1-2
 - applying during conversion of a physical standby database to a logical standby database, 4-5
 - archiving on the standby system, 1-4, 6-1
 - building a dictionary in, 4-4
 - manually transferring, 2-5
 - transmitting, 1-2, 1-3, 5-1 to 5-18
 - validated, 1-10
- redo log files
 - delaying application, 6-3
- redo logs
 - automatic application on physical standby databases, 6-4
 - configuration considerations, 2-9
 - in Data Guard configurations, 2-8
 - secure redo transmission, 5-13
 - update standby database tables, 1-10
- redo transport services, 5-1 to 5-30
 - archive destinations
 - alternate, A-3
 - archived redo log repository, 5-2
 - Oracle Change Data Capture, 5-3
 - Oracle Streams, 5-2
 - quotas, D-5
 - re-archiving to failed destinations, 5-16, 14-22
 - role-specific, 5-14
 - shared, 5-24, 14-10
 - specifying with the LOG_ARCHIVE_DEST_n initialization parameter, 5-4
 - archived redo log files
 - generating filenames, 5-22
 - listing with V\$ARCHIVED_LOG view, 5-22
 - specifying directory locations for, 5-21
 - defined, 1-3, 5-1
 - handling archive failures, 5-16, 14-22

- monitoring, 5-29, 5-30
- network
 - ASYNC network transmission, 5-10
 - SYNC network transmission, 5-10
 - tuning, A-9
- protection modes
 - choosing, 5-18
 - maximum availability mode, 1-7, 5-18
 - maximum performance mode, 1-8, 5-19
 - maximum protection mode, 1-7, 5-18
 - providing no data loss, 5-10
 - setting, 5-19
- secure redo transmission, 5-13
- standby redo log files
 - configuring, 3-2
- synchronous and asynchronous disk I/O, 14-2
- using the log writer process, 5-12, 14-6, 14-23
- REGISTER LOGFILE clause, 5-28
 - of ALTER DATABASE, 15-3, A-4
 - registering missing log files, 5-28
- REGISTER LOGICAL LOGFILE clause, 12-19
 - of ALTER DATABASE, 5-28, 7-16, 12-19
- registering
 - archived redo log files, 5-28, 12-19
 - during failover, 7-16
 - missing log files, 5-28
 - partial archived redo log files, 12-19
- RELY constraint
 - creating, 4-2
- remote file server process (RFS)
 - defined, 2-8, 5-11, 6-1
 - log writer process and, 6-2
 - standby redo log files reused by, 5-23
- REMOTE_LOGIN_PASSWORDFILE initialization
 - parameter
 - secure redo transmission, 5-14
- RENAME FILE clause
 - of ALTER DATABASE, A-2
- renaming
 - datafiles
 - on the primary database, 8-11
 - setting the STANDBY_FILE_MANAGEMENT parameter, 8-10
- REOPEN attribute, 14-22
 - LOG_ARCHIVE_DEST_n initialization
 - parameter, 5-16
- reporting operations
 - configuring, 1-2
 - offloading on the standby database, 1-10
 - performing on a logical standby database, 1-2
- repository
 - for temporary storage of archived redo log files, 5-2
- requirements
 - data protection modes, 5-19
 - of a rolling upgrade, 11-2
- RESET attribute
 - LOG_ARCHIVE_DEST_STATE_n initialization
 - parameter, 5-3
- RESETLOGS_ID columns
 - viewing in V\$DATABASE_INCARNATION view, 8-17
- resolving
 - logical corruptions, 1-10
- restart considerations
 - SQL Apply, 9-3
- restore points
 - creating, 12-30
 - creating guaranteed, 12-30
 - using the CREATE RESTORE POINT command, 12-30
- resynchronizing
 - logical standby databases with a new branch of redo, 9-20
 - physical standby databases with a new branch of redo, 8-13
- retrieving
 - missing archived redo log files, 1-3, 1-10, 5-25, 12-18
- reusing records in the control files, 5-23
- RFS
 - See* remote file server process (RFS)
- RMAN
 - incremental backup
 - rolling forward databases that lag far behind the primary, 12-33
 - rolling forward databases when nologging changes applied to a small subset of data files, 12-34
 - rolling forward databases when nologging changes are widespread, 12-36
 - incremental backups, 12-33
 - rolling forward physical standby databases, 12-33
- RMAN BACKUP INCREMENTAL FROM SCN
 - command, 12-33
- role management services
 - defined, 7-1
- role transitions, 1-5, 7-1
 - and cascaded configurations, E-4
 - choosing a type of, 7-2
 - choosing the best available standby database, 12-9
 - defined, 1-5
 - flashing back the databases after, 7-20
 - logical standby database and, 7-13
 - monitoring, 8-14
 - physical standby databases and, 7-7
 - reversals, 1-5, 7-2
 - writing triggers to manage tasks after, 7-5, 7-7
- role-based destinations
 - setting, 14-26
- rollback
 - after switchover failures, A-7
- rolling upgrade
 - software requirements, 2-5
- rolling upgrades
 - benefits, 11-1
 - patch set releases, 2-5
 - requirements, 11-2

- setting the COMPATIBLE initialization parameter, 11-2, 11-3, 11-10
- unsupported data types and storage attributes, 11-3

ROWID data types

- in logical standby databases, C-2

S

scenarios

- cascaded destinations, E-4 to E-7
- choosing best standby database for role transition, 12-9
- recovering
 - a logical standby database, 12-41
 - after NOLOGGING is specified, 12-40
 - from a network failure, 12-39
- time lag in redo logs, 12-37

schemas

- data manipulation on logical standby databases, 1-10
- DBA_LOGSTDBY_SKIP list of skipped, C-3
- identical to primary database, 1-2
- skipped by SQL Apply, C-3

SCN

- determine the highest applicable (newest), 12-17
- using for incremental backups, 12-33

secure redo transmission, 5-13

segment compression storage type

- in logical standby databases, C-2

sequences

- unsupported on logical standby databases, C-3

SERVICE attribute, 14-12

SET STANDBY DATABASE clause

- of ALTER DATA, 15-3
- of ALTER DATABASE, 7-6, 15-3

setting

- data protection mode, 5-19
- VALID_FOR attributes
 - for a logical standby database, 12-3
 - for a physical standby database, 12-1

sharing

- destinations, 5-24, 14-10
- flash recovery area, 5-7

shutting down

- physical standby database, 8-2

simulating

- standby database environment, 2-5

SKIP procedure

- of DBMS_LOGSTDBY, A-9

SKIP_ERROR procedure

- of the DBMS_LOGSTDBY package, A-4

SKIP_TRANSACTION procedure

- of DBMS_LOGSTDBY, A-9

snapshot

- obtained by Flashback Query, 4-4

software requirements, 2-5

- Oracle Database Enterprise Edition, 2-5
- rolling upgrades, 2-5

Spatial data types

- in logical standby databases, C-2

SQL Apply, 6-5, 9-3

- after an OPEN RESETLOGS, 9-20

- ANALYZER process, 9-2

- APPLIER process, 9-2

- applying CREATE TABLE AS SELECT (CTAS) statements, 9-4

- applying DDL transactions, 9-4

- applying DML transactions, 9-4

- architecture, 9-1, 9-10

- BUILDER process, 9-2

- COORDINATOR process, 9-2

- defined, 1-4, 6-2

- deleting archived redo log files, 9-12

- parallel DDL transactions, 9-4

- parallel DML (PDML) transactions, 9-4

- performing a rolling upgrade, 11-1

- PREPARER process, 9-2

- READER process, 9-2

- requirements for rolling upgrades, 11-2

- restart considerations, 9-3

- role transitions and cascaded configurations, E-4

- rolling upgrades, 2-5

- schemas skipped, C-3

- starting

- real-time apply, 6-5

- stopping

- real-time apply, 6-5

- support for DDL statements, C-1

- support for PL/SQL supplied packages, C-3

- support for storage types, C-2

- supported data types, C-1

- transaction size considerations, 9-2

- unsupported data types, C-2

- unsupported PL/SQL supplied packages, C-3

- unsupported storage types, C-2

- viewing current activity, 9-2

- of processes, 9-2

- what to do if it stops, A-8

- with real-time apply, 9-14

SQL Apply resolving archive gaps, 5-27

SQL sessions

- causing switchover failures, A-5

SQL statements

- executing on logical standby databases, 1-2, 1-4

- skipping on logical standby databases, C-4

- switchover and, 7-8

standby database

- creating logical, 4-1

standby databases

- about creating using RMAN, F-2

- applying redo data on, 6-1

- applying redo log files on, 1-4, 1-10

- cascading, E-1

- choosing a target for role transitions, 12-9

- configuring, 1-1

- archived redo log repository, 5-2

- cross-instance archival, D-3

- delayed application of archived redo log files on, 12-37

- mandatory destinations, 5-22
- maximum number of, 2-1
- on Real Application Clusters, 1-2, D-1, D-3
- on remote locations, 1-2
- optional destinations, 5-22
- single-instance, 1-2
- creating, 1-2, 3-1, F-15
 - checklist of tasks, 4-3
 - directory structure considerations, 2-6
 - if primary uses ASM or OMF, 12-48
 - on local host, F-15
 - on remote host with different directory structure, F-10
 - on remote host with same directory structure, F-9
 - using RMAN, F-5
 - with a time lag, 6-3, 12-37
- creating control files
 - using RMAN, F-2
- DB_FILE_NAME_CONVERT initialization
 - parameter, F-4
- defined, 2-1
- failover
 - preparing for, 7-6
- failover to, 7-6
 - re-creating after, 7-10
- hardware requirements, 2-4
- log apply services on, 6-1
- LOG_FILE_NAME_CONVERT initialization
 - parameter, F-5
- modifying the control file, 8-10
- naming datafiles using RMAN, F-3
- operational requirements, 2-4, 2-5
- preparing to use RMAN, F-1
- recovering through OPEN RESETLOGS, 8-13
- resynchronizing with the primary database, 1-10
- reverting back to primary database, A-7
- rolling forward with RMAN incremental backups, 12-33
- SET AUXNAME command, F-5
- SET NEWNAME command, F-4, F-5
- setting the NETWORK_TIMEOUT attribute, 5-13
- setting up to use OMF files, F-7, F-8
- sharing
 - a flash recovery area, 5-7
- software requirements, 2-5
- starting log apply services on physical, 6-4
- starting RMAN and standby instance, F-7
- viewing database incarnation information, 8-17
- viewing RESETLOGS_ID, 8-17
- wait events configured for, 5-30
- See also* logical standby databases
- See also* physical standby databases
- standby redo log files
 - advantages, 2-9
 - and real-time apply, 2-8, 6-2
 - applying
 - to standby databases, 1-3
 - creating, 3-2, 3-3
 - log groups and members, 3-2
 - cross-instance archival and, D-3
 - network transmission modes for, 5-10
 - on raw devices, 2-9
 - overview, 2-9
 - Real Application Clusters and, 3-3, D-3
 - requirements for
 - cascaded destinations, 2-9
 - maximum availability mode, 2-9
 - maximum protection mode, 2-9
 - protection modes, 1-8
 - real-time apply, 2-9
- standby redo log groups
 - adding members, 5-23
 - determining if you have enough, 5-23
 - recommended number, 3-2
- standby role, 1-2
- STANDBY_ARCHIVE_DEST initialization
 - parameter, 5-21
 - archiving to the recovery area, 5-7
 - implicit default value, 5-21
- STANDBY_FILE_MANAGEMENT initialization
 - parameter
 - setting for transportable tablespaces, 8-10
 - when renaming datafiles, 8-10
- START LOGICAL STANDBY APPLY clause
 - IMMEDIATE keyword, 6-5, 9-14
 - of ALTER DATABASE, 4-7, 6-5, 7-18, 11-6, A-9
- starting
 - logical standby databases, 4-7
 - physical standby databases, 3-11
 - real-time apply, 6-5
 - on logical standby databases, 6-5, 9-14
 - on physical standby databases, 6-5
 - Redo Apply, 3-11, 6-5, 8-1
 - SQL Apply, 4-7, 6-5
- STOP LOGICAL STANDBY APPLY clause
 - of ALTER DATABASE, 6-5, 7-17, 11-6, 12-19, 15-4
- stopping
 - real-time apply
 - on logical standby databases, 6-5
 - real-time apply on physical standby databases, 6-5
 - Redo Apply, 6-5
 - SQL Apply, 6-5
- storage attributes
 - unsupported during a rolling upgrade, 11-3
- storage types
 - cluster tables, C-2
 - heap-organized tables, C-2
 - in logical standby databases, C-2
 - index-organized tables, C-2
 - segment compression, C-2
 - supported, C-2
 - unsupported, C-2
- supplemental logging
 - setting up to log primary key and unique-index columns, 4-4, 9-4
- supported data types
 - for logical standby databases, C-1, C-5
- supported PL/SQL supplied packages, C-3

- supported storage types, C-2
- SWITCH LOGFILE clause
 - of ALTER SYSTEM, 3-12
- SWITCHOVER_STATUS column
 - of V\$DATABASE view, 7-7, 7-8, A-4
- switchovers, 1-5
 - and cascaded configurations, E-4
 - Redo Apply, E-4
 - SQL Apply, E-4
 - choosing a target standby database, 7-3
 - control files and, 7-8
 - defined, 1-5, 7-2
 - displaying history with DBA_LOGSTDBY_
 - HISTORY, 16-1
 - fails with ORA-01102, A-6
 - flashing back databases after, 7-20
 - initiating on the primary database, 7-8
 - logical standby databases and, 7-13
 - manual versus automatic, 1-5, 7-1
 - monitoring, 8-14
 - no data loss and, 7-2
 - physical standby databases and, 7-5, 7-8
 - preparing for, 7-5
 - prevented by
 - active SQL sessions, A-5
 - active user sessions, A-6
 - CJQ0 process, A-6
 - DBSNMP process, A-6
 - processes, A-6
 - QMN0 process, A-6
 - seeing if the last archived redo log file was
 - transmitted, A-4
 - simplifying with Data Guard broker, 1-6, 7-1
 - SQL statements and, 7-8
 - standby databases not involved in, 7-9
 - starting over, A-7
 - typical use for, 7-4
 - using DB_ROLE_CHANGE system event
 - after, 7-5, 7-7
 - using Real Application Clusters and, D-5, D-6
 - V\$DATABASE view and, 7-7, 7-8
 - verifying, 7-8
- SYNC attribute, 14-23
 - LOG_ARCHIVE_DEST_n initialization
 - parameter, 5-10
 - setting for data protection, 5-19
- SYS user account
 - password file requirements, 5-13
 - REMOTE_LOGIN_PASSWORDFILE initialization
 - parameter, 5-14
- system events
 - writing triggers for DB_ROLE_CHANGE, 7-5, 7-7
- system global area (SGA)
 - logical change records staged in, 9-2
- system resources
 - efficient utilization of, 1-10

T

- tables
 - logical standby databases
 - adding on, 9-18
 - re-creating tables on, 9-18
 - unsupported on, C-3
 - unsupported in a logical standby database, 11-3
- tablespaces
 - adding
 - a new datafile, A-12
 - to primary database, 8-6
 - dropping from primary database, 8-9
 - monitoring status changes, 8-14
 - moving between databases, 8-10
- target standby database
 - for switchover, 7-3
- target standby databases
 - choosing a logical standby database for
 - failover, 12-16
 - choosing physical standby databases for
 - failover, 12-10
- TEMPLATE attribute, 14-24
- terminating
 - network connection, 14-19
- text indexes
 - supported by logical standby databases, C-2
- throughput
 - on logical standby databases, 9-4
- time lag
 - delaying application of archived redo log
 - files, 6-3, 12-37, 14-8
 - in standby database, 6-3, 12-37, 14-8
- TIMESTAMP data types
 - in logical standby databases, C-2
- tnsnames.ora file
 - redo transport services tuning and, A-9
 - troubleshooting, 12-40, A-2, A-7, A-9
- trace files
 - levels of tracing data, G-2
 - location of, G-1
 - setting, G-1
 - tracking real-time apply, G-2
- transaction size considerations
 - SQL Apply, 9-2
- transportable tablespaces
 - defining location with DB_FILE_NAME_
 - CONVERT parameter, 8-10
 - setting the STANDBY_FILE_MANAGEMENT
 - parameter, 8-10
 - using with a physical standby database, 8-10
- triggers
 - handled on a logical standby database, 9-20
- troubleshooting
 - if SQL Apply stops, A-8
 - last redo data was not transmitted, A-4
 - listener.ora file, 12-40, A-2, A-9
 - logical standby database failures, A-4
 - processes that prevent switchover, A-6
 - SQL Apply, A-8
 - switchovers, A-4

- active SQL sessions, A-5
- active use sessions, A-6
- ORA-01102 message, A-6
- roll back and start over, A-7
- tnsnames.ora file, 12-40, A-2, A-7, A-9
- tuning
 - determining if you have enough standby redo log groups, 5-23
 - log apply rate for Redo Apply, 8-22

U

- UNDO_RETENTION initialization parameter
 - recommendations, 4-4
- unique-index columns
 - logged with supplemental logging, 4-4, 9-4
- unrecoverable operations, 12-41
 - backing up after, 12-43
- unsupported data types
 - during a rolling upgrade, 11-3
- unsupported PL/SQL supplied packages, C-3
- unsupported storage types, C-2
- unsupported tables
 - for logical standby database during a rolling upgrade, 11-3
- upgrading
 - Oracle Database, B-1
 - Oracle Database software, 11-1
 - Oracle database software, 2-5
 - Oracle database software version, 11-1
 - requirements, 11-2
- UROWID data types
 - in logical standby databases, C-2
- user errors
 - safeguarding against, 1-9
- user sessions
 - causing switchover failures, A-6
- USER_DUMP_DEST initialization parameter, G-1
- user-defined data types
 - in logical standby databases, C-2
- USING CURRENT LOGFILE clause, 8-5
 - starting real time apply, 6-5
- using RMAN
 - using image copies, F-15

V

- V\$ARCHIVE_DEST view, 5-29, 12-40, 16-1, A-2
 - displaying implicit default value of STANDBY_ARCHIVE_DEST parameter, 5-21
 - displaying information for all destinations, 16-1
- V\$ARCHIVE_DEST_STATUS view, 5-29, 8-20, 16-1
 - log apply services and MANAGED REAL TIME column, 8-20
- V\$ARCHIVE_GAP view, 16-2
 - finding gaps on physical standby databases, 5-27
- V\$ARCHIVED_LOG view, 5-22, 5-29, 8-20, 12-45, 16-2, A-4
 - determining the most recently archived redo log file, 5-29
 - locating missing log files, 5-27
- V\$DATABASE view, 16-2
 - monitoring fast-start failover, 8-14
 - switchover and, 7-7, 7-8
 - SWITCHOVER_STATUS column and, 7-7, 7-8, A-4
- V\$DATABASE_INCARNATION view, 16-2
 - obtaining database incarnation information, 8-17
- V\$DATAFILE view, 12-41, 12-43, 16-2
- V\$DATAGUARD_CONFIG view, 16-2
 - listing database names defined with LOG_ARCHIVE_CONFIG, 16-2
- V\$DATAGUARD_STATS view, 16-2
- V\$DATAGUARD_STATUS view, 8-21, 16-2
- V\$LOG view, 5-29, 16-2
- V\$LOG_HISTORY view, 8-18, 8-20, 16-2
- V\$LOGFILE view, 16-2
- V\$LOGSTDBY view, 1-xxiii
- V\$LOGSTDBY_PROCESS view, 9-2, 9-6, 9-7, 9-11, 9-23, 9-24, 16-2
- V\$LOGSTDBY_PROGRESS view, 9-8, 16-2
 - querying SCN information and, 12-17
 - RESTART_SCN column, 9-3
- V\$LOGSTDBY_STATE view, 7-3, 9-9, 9-11, 16-2
- V\$LOGSTDBY_STATS view, 9-2, 9-9, 16-2
 - failover characteristics, 9-6
- V\$LOGSTDBY_TRANSACTION view, 16-3
- V\$MANAGED_STANDBY view, 8-19, 16-3
- V\$RECOVER_FILE view, 8-14
- V\$SESSION view, A-5, A-6
- V\$STANDBY_LOG view, 7-19, 16-3
- V\$THREAD view, 8-14
- VALID_FOR attribute, 14-26
 - examples, 5-15
 - overview, 5-14
 - values and role-based conditions, 14-27
 - verifying, 12-9
- validating
 - redo data, 1-10
- VARCHAR data types
 - in logical standby databases, C-2
- VARCHAR2 data types
 - in logical standby databases, C-2
- VERIFY attribute, 14-28
- verifying
 - contents of archived redo log files, 14-28
 - logical standby databases, 4-7
 - physical standby databases, 3-12
 - role-based destination settings, 12-9
 - standby redo log groups, 3-4
- versions
 - upgrading Oracle database software, 11-1
- View, 9-8, 9-9
- views, 8-15, 16-1
 - DBA_LOGSTDBY_EVENTS, 9-5, 16-1, A-9
 - DBA_LOGSTDBY_HISTORY, 16-1
 - DBA_LOGSTDBY_LOG, 9-5, 12-17, 16-1
 - DBA_LOGSTDBY_NOT_UNIQUE, 16-1
 - DBA_LOGSTDBY_PARAMETERS, 16-1
 - DBA_LOGSTDBY_SKIP, 16-1

DBA_LOGSTDBY_SKIP_TRANSACTION, 16-1
DBA_LOGSTDBY_UNSUPPORTED, 16-1, C-3
DBA_TABLESPACES, 8-14
displaying history of switchovers and
failovers, 16-1
GV\$INSTANCE, D-6
V\$ARCHIVE_DEST, 5-29, 12-40, 16-1, A-2
V\$ARCHIVE_DEST_STATUS, 5-29, 8-20, 16-1
V\$ARCHIVE_GAP, 16-2
V\$ARCHIVED_LOG, 5-22, 5-29, 8-20, 12-45, 16-2
V\$DATABASE, 16-2
V\$DATABASE_INCARNATION, 16-2
V\$DATAFILE, 12-41, 12-43, 16-2
V\$DATAGUARD_CONFIG, 16-2
V\$DATAGUARD_STATS, 16-2
V\$DATAGUARD_STATUS, 8-21, 16-2
V\$LOG, 5-29, 16-2
V\$LOG_HISTORY, 8-18, 8-20, 16-2
V\$LOGFILE, 16-2
V\$LOGSTDBY, 1-xxiii
V\$LOGSTDBY_PROCESS, 9-2, 9-6, 16-2
V\$LOGSTDBY_PROGRESS, 9-8, 16-2
V\$LOGSTDBY_STATE, 9-9, 16-2
V\$LOGSTDBY_STATS, 9-2, 9-9, 16-2
V\$LOGSTDBY_TRANSACTION, 16-3
V\$MANAGED_STANDBY, 8-19, 16-3
V\$RECOVER_FILE, 8-14
V\$SESSION, A-5, A-6
V\$STANDBY_LOG, 3-4, 7-19, 16-3
V\$THREAD, 8-14

W

wait events, 5-30
for standby destinations, 5-30
monitoring the performance of the redo transport
modes, 5-30
WAITING FOR DICTIONARY LOGS state, 9-11
waiting on gap state, 9-12

X

XMLType data types
in logical standby databases, C-2

Z

zero data loss
See no data loss
zero downtime instantiation
logical standby databases, 4-3

