Oracle® Data Guard

Concepts and Administration

10*g* Release 1 (10.1)

Part No. B10823-01

December 2003

This guide describes Oracle Data Guard concepts and helps you implement and manage standby databases to ensure high availability, data protection, and disaster recovery for enterprise data.

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

Part No. B10823-01

Primary Author: Viv Schupmann

Contributing Authors: Lance Ashdown

Contributors: Rick Anderson, Cathy Baird, Tammy Bednar, Anand Beldalker, Barbara Benton, Lucy Burgess, Larry Carpenter, Wei Chen, Laurence Clarke, Rhonda Day, Jeff Detjen, Ray Dutcher, Chuck Freiwald, Mahesh Girkar, Roshan Gupta, Ray Guzman, Susan Hillson, Mark Johnson, Sadhana Kyathappala, Steve Lee, Steve McGee, Bob McGuirk, Jeff Nesheiwat, Muthu Olagappan, Deborah Owens, Ashish Ray, Charles Sondey, Ingrid Stuart, Lawrence To, Mike Smith, Randy Urbano, Ric Van Dyke, Lik Wong

Contents

4 Creating a Logical Standby Database

5 Log Transport Services

List of Examples

List of Figures

List of Tables

Send Us Your Comments

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

Part No. B10823-01

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

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

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

- Electronic mail: nedc-doc_us@oracle.com
- FAX: 603.897.3825 Attn: Oracle Data Guard Documentation
- Postal service:
  Oracle Corporation
  Oracle Data Guard Documentation
  One Oracle Drive
  Nashua, NH 03062-2804
  U.S.A.

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

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

Preface

Oracle 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.

This preface contains the following topics:

- Audience
- Documentation Accessibility
- Organization
- Related Documentation
- Conventions

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. Standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For additional information, visit the Oracle Accessibility Program Web site at

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

Accessibility of Code Examples in Documentation JAWS, a Windows screen reader, may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, JAWS may not always read a line of text that consists solely of a bracket or brace.

Organization

This document contains:

Part I, "Concepts and Administration"

Chapter 1, "Introduction to Oracle Data Guard"

This chapter offers a general overview of the Oracle Data Guard architecture.

Chapter 2, "Getting Started with Data Guard"

This chapter describes physical and logical databases in more detail and the various interfaces you can use to manage the Data Guard configuration. It also describes the operational requirements for using Data Guard and provides recommendations for setting up directory structures on standby databases.

Chapter 3, "Creating a Physical Standby Database"

This chapter explains how to create a physical standby database.

Chapter 4, "Creating a Logical Standby Database"

This chapter explains how to create a logical standby database.

Chapter 5, "Log Transport Services"

This chapter introduces log transport services. It describes the data protection modes that protect the production database against loss in the event of an unplanned outage, and it provides procedures and guidelines for configuring log transport services on a primary and standby database.

Chapter 6, "Log Apply Services"

This chapter introduces log apply services. It provides guidelines for managing log apply services for physical and logical standby databases.

Chapter 7, "Role Management"

This chapter introduces role management services. It provides information about database failover and switchover role transitions.

Chapter 8, "Managing a Physical Standby Database"

This chapter describes how to manage a physical standby database. It provides information about monitoring and responding to events that affect a standby database.

Chapter 9, "Managing a Logical Standby Database"

This chapter describes how to manage a logical standby database. It provides information about managing SQL Apply, system tuning, and tablespace management.

Chapter 10, "Data Guard Scenarios"

This chapter describes common database scenarios such as creating, recovering, failing over, switching over, configuring, and backing up standby and primary databases.

Part II, "Reference"

Chapter 11, "Initialization Parameters"

This reference chapter describes initialization parameters for each Oracle instance, including the primary database and each standby database in the Data Guard environment.

Chapter 12, "LOG_ARCHIVE_DEST_n Parameter Attributes"

This reference chapter provides syntax and examples for the attributes of the `LOG_ARCHIVE_DEST_`*n* initialization parameter.

Chapter 13, "SQL Statements Relevant to Data Guard"

This reference chapter provides SQL statements that are useful for performing operations on a Data Guard configuration.

Chapter 14, "Views Relevant to Oracle Data Guard"

This reference chapter lists views that contain useful information for monitoring the Data Guard environment. It summarizes the columns contained in each view and provides a description for each column.

Part III, "Appendixes and Glossary"

Appendix A, "Troubleshooting Data Guard"

This appendix discusses troubleshooting tips for Data Guard and standby databases.

Appendix B, "Data Guard and Real Application Clusters"

This appendix describes the primary and standby database configurations in a Real Application Clusters environment.

Appendix C, "Cascaded Redo Log Destinations"

This appendix describes how to implement cascaded redo log file destinations, whereby a standby database receives redo data from another standby database, instead of directly from the primary database.

Appendix D, "Creating a Physical Standby Database with Recovery Manager"

This appendix describes how to use Recovery Manager to create a physical standby database.

Appendix E, "Setting Archive Tracing"

This appendix describes how the `LOG_ARCHIVE_TRACE` parameter controls output generated by the ARC*n*, LGWR, and foreground processes on the primary database, and the RFS and FAL server processes on the standby database.

Appendix F, "Sample Disaster Recovery ReadMe File"

This appendix provides a sample ReadMe file that includes the kind of information that the person who is making disaster recovery decisions would need when deciding which standby database should be the target of the failover operation.

Related Documentation

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.

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 Net Services Administrator's Guide*
- *SQL*Plus User's Guide and Reference*
- *Oracle High Availability Architecture and Best Practices*

If you need to upgrade existing Data Guard configurations to this Oracle release, see *Oracle Database Upgrade Guide* for complete instructions. In addition, refer to *Oracle Database Concepts* for information about other Oracle products and features that provide disaster recovery and high-availability solutions.

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 log 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.

Printed documentation is available for sale in the Oracle Store at

`http://oraclestore.oracle.com/`

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

`http://otn.oracle.com/membership/`

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

`http://otn.oracle.com/documentation/`

Conventions

This section describes the conventions used in the text and code examples of this document. The following table describes those conventions and provides examples of their use.

Convention	Meaning	Example
[ ]	Brackets enclose one or more optional items. Do not enter the brackets.	`DECIMAL (`*`digits`*` [ , `*`precision`*` ])`
{ }	Braces enclose two or more items, one of which is required. Do not enter the braces.	`{ENABLE | DISABLE}`
|	A vertical bar represents a choice of two or more options within brackets or braces. Enter one of the options. Do not enter the vertical bar.	`{ENABLE | DISABLE}` `[COMPRESS | NOCOMPRESS]`
...	Horizontal ellipsis points indicate either:	
	■ That we have omitted parts of the code that are not directly related to the example	`CREATE TABLE ... AS `*`subquery`*`;`
	■ That you can repeat a portion of the code	`SELECT `*`col1`*`, `*`col2`*`, ... , `*`coln`*` FROM employees;`
. . .	Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.	

Convention	Meaning	Example
Bold	Bold typeface indicates terms that are defined in the text or terms that appear in a glossary, or both.	When you specify this clause, you create an **index-organized table**.
`UPPERCASE monospace (fixed-width font)`	Uppercase monospace typeface indicates elements supplied by the system.	You can back up the database by using the `BACKUP` command. Use the `DBMS_STATS.GENERATE_STATS` procedure.
`lowercase monospace (fixed-width font)`	Lowercase monospace typeface indicates executables, filenames, directory names, and sample user-supplied elements.	Enter `sqlplus` to open SQL*Plus. Back up the datafiles and control files in the `/disk1/oracle/dbs` directory. The `department_id`, `department_name`, and `location_id` columns are in the `hr.departments` table.
`lowercase monospace (fixed-width font) italic`	Lowercase monospace italic font represents placeholders or variables.	You can specify the *`parallel_clause`*. Run `U`*`old_release`*`.SQL` where *`old_release`* refers to the release you installed prior to upgrading.
`MixedCase monospace (fixed-width font)`	Mixed-case monospace typeface indicates a Data Guard broker database property. The mixed case helps you visually differentiate a Data Guard broker property from other system-supplied elements, which are always shown in uppercase typeface. Mixed-case monospace typeface can also indicate other programmatic elements. Enter these elements as shown.	The `StandbyFileManagement` property corresponds to the `STANDBY_FILE_MANAGEMENT` initialization parameter. The `JRepUtil` class implements these methods.

What's New in Oracle Data Guard?

The features and enhancements described in this preface were added to Oracle Data Guard in Release 10.1. The new features are described under the following main areas:

- New Features Common to Physical and Logical Standby Databases
- New Features Specific to Physical Standby Databases
- New Features Specific to Logical Standby Databases

New Features Common to Physical and Logical Standby Databases

The following enhancements to Oracle Data Guard in Release 10.1 improve ease-of-use, manageability, performance, and include innovations that improve disaster recovery capabilities:

- **Real-Time Apply**

  Data Guard log apply services can now apply redo data as it is received on the logical or physical standby database, without waiting for the current standby online redo log file to be archived. This allows reporting on more up-to-date data, quicker failovers and switchovers, and reduces planned and unplanned downtime.

  See Also: Chapter 6, "Log Apply Services"

- **Recovery Through Open Resetlogs**

  Data Guard supports recovery through open resetlogs by allowing recovery on a standby database to react appropriately to a `RESETLOGS` operation, instead of requiring the standby database to be re-created. Because an `ALTER DATABASE OPEN RESETLOGS` statement is always issued after a point-in-time recovery or a Flashback Database operation, the recovery through resetlogs feature allows the standby database to resume.

See Also: Section 10.4, "Using Flashback Database After Issuing an Open Resetlogs Statement"

- **Flashback Database Support**

  Data Guard supports the new Flashback Database feature that allows a standby database to be quickly and easily flashed back to an arbitrary point in time. This feature provides the following benefits when used with Data Guard:

  – Flashback Database removes the need to re-create the primary database after a failover. After a failover, the original primary database can now be flashed back to a point in time before the failover and converted into a standby database. Once all of the logs have been applied, the original primary database can be switched back to the primary role.

  – Provides an alternative to delaying the application of redo data to protect against user errors or logical corruptions. Therefore, standby databases can be more closely synchronized with the primary database, reducing failover and switchover times.

  See Also: See Section 6.2.2 and the *Oracle Database Backup and Recovery Advanced User's Guide* for more information about using Flashback Database

- **Improved Redo Data Transmission Security**

  Data Guard log transport services now use authenticated network sessions to transfer redo data among the members of a Data Guard configuration. If the Oracle Advanced Security option is installed, security can be increased still further by using encryption and integrity checksums on network transmission of redo data.

  See Also: Section 5.3.3, "Providing for Secure Redo Data Transmission", Chapter 12, "LOG_ARCHIVE_DEST_n Parameter Attributes", and *Oracle Advanced Security Administrator's Guide*

- **Improved Data Guard Support for Real Application Clusters**

  The Data Guard broker can now be used to manage Data Guard configurations that contain Real Application Clusters primary or standby databases.

  See Also: *Oracle Data Guard Broker*

- **Dynamically Add Standby Databases to Real Applications Clusters**

  It is now 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, without shutting down the primary database.

  This enhancement requires using initialization parameters that are new in Oracle Database 10*g* Release 1:

  – `DB_UNIQUE_NAME`

  – `LOG_ARCHIVE_CONFIG`

  See Also: Chapter 5, "Log Transport Services" and Chapter 12, "LOG_ARCHIVE_DEST_n Parameter Attributes"

- **Simplified Data Guard Configuration Management**

  The new `VALID_FOR` attribute of the `LOG_ARCHIVE_DEST_`*n* initialization parameter can be used to create initialization parameter files that are role independent. This simplifies switchovers and failovers because it is no longer necessary to enable and disable role-specific archiving destinations after a performing a database role transition. This feature makes it possible to use the same parameter file whether the database is running in the primary or the standby role.

  See Also: Chapter 5, "Log Transport Services" and Chapter 12, "LOG_ARCHIVE_DEST_n Parameter Attributes"

- **Automated Disk-Based Backup and Flashback Recovery**

  Managing files needed for backup and recovery is simplified when these files are stored in a flash recovery area. If a flash recovery area is configured, Data Guard will implicitly set the `LOG_ARCHIVE_DEST_10` initialization parameter to point to the flash recovery area and will use this destination for local archiving unless you explicitly configure another local archive destination.

See Also: Chapter 5, "Log Transport Services" and *Oracle Database Backup and Recovery Basics*

- **Archiver Process Supports Remote Standby Redo Log Files**

  The archiver process (ARC*n*) can now transmit redo data to remote destinations that are configured to use standby redo log files. Failovers are simplified when this feature is used, because partially filled archived redo log files no longer have to be registered before performing a failover.

  See Also: Chapter 5, "Log Transport Services"

- **Change In Default Archival Behavior**

  The default archival processing in a Data Guard configuration has changed so that archiver processes (ARC*n*) on the primary database will completely and successfully archive the local online redo log files before transmitting the redo data to remote standby destinations. Before release 10.1, the default behavior was to transmit redo data to the standby destination at the same time the online redo log file was being archived to the local online redo log files. This change allows the redo log groups to be reused faster and reduces the likelihood of a database hang due to lack of available redo log groups.

  See Also: Section 5.3.1, "Using Archiver Processes (ARCn) to Archive Redo Data"

- **Simplified Management of Redo Log Archiving**

  Automatic archiving is now enabled by default when a database is put into archivelog mode.

  See Also: Chapter 5, "Log Transport Services" and *Oracle Database Administrator's Guide*

- **Secure Redo Transmission**

  Log transport services now 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.

See Also: Section 5.3.3, "Providing for Secure Redo Data Transmission" and the *Oracle Advanced Security Administrator's Guide*

New Features Specific to Physical Standby Databases

The following list summarizes the new features that are specific to physical standby databases in Oracle Database 10*g*:

- **New Default Behavior for the `STARTUP`, `MOUNT`, and `OPEN` Statements**

  The `STARTUP`, `MOUNT`, and `OPEN` statements have new default behaviors that simplify their use with a physical standby database:

  – The `STARTUP` command will now start, mount and open a physical standby database in read-only mode in a single step.

  – The `STANDBY DATABASE` keywords are now optional when using the `ALTER DATABASE MOUNT` statement to mount a physical standby database.

  – The `READ ONLY` keywords are now optional when using the `ALTER DATABASE OPEN` statement to open a physical standby database.

  See Also: Chapter 13, "SQL Statements Relevant to Data Guard" and the *Oracle Database SQL Reference*

New Features Specific to Logical Standby Databases

Logical standby databases, first released with Oracle Database in Release 9.2, were enhanced in this release to allow rolling upgrades, improve the overall ease-of-use and manageability, expand the disaster recovery capabilities, and simplify the steps to create a logical standby database. The following list summarizes the new features for logical standby databases in Oracle Database 10*g*:

- **Zero Downtime Instantiation**

  It is now possible to create a logical standby database without having to shut down or quiesce the primary database. This is achieved by using an online backup of the primary database and creating a logical standby control file.

  See Also: Chapter 4, "Creating a Logical Standby Database"

- **Rolling Database Upgrades with SQL Apply**

  In a future patchset release of Oracle Database 10*g*, it will be possible to do a rolling upgrade using logical standby databases. The foundation for rolling upgrades is now implemented into the SQL Apply technology so that the primary database incurs minimal downtime when you upgrade Oracle Database software on each database in the Data Guard configuration. For example, using SQL Apply and logical standby databases, you will be able to upgrade the Oracle Database software from patchset release 10.1.0.*n* to the next database 10.1.0.(*n+1*) patchset release.

  > **See Also:** Section 9.2 and the ReadMe file for the applicable Oracle Database 10*g* patchset release

- **Support for Maximum Protection Mode**

  With the introduction of support for standby redo log files, it is now possible to have a logical standby database be part of a Data Guard configuration running in maximum protection mode.

  > **See Also:** Section 5.6, "Setting Up a Data Protection Mode"

- **Support for Additional Datatypes**

  Logical standby databases now include support for `LONG`, `LONG RAW`, and `NCLOB` data types. Also, support for index organized tables was added provided the index organized table does not contain either an overflow segment or any LOB column.

  > **See Also:** Section 4.1.1, "Determine Support for Datatypes and Storage Attributes for Tables"

Part I

Concepts and Administration

This part contains the following chapters:

1 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 by recovering the redo data received from the primary database.

- **Logical standby database**

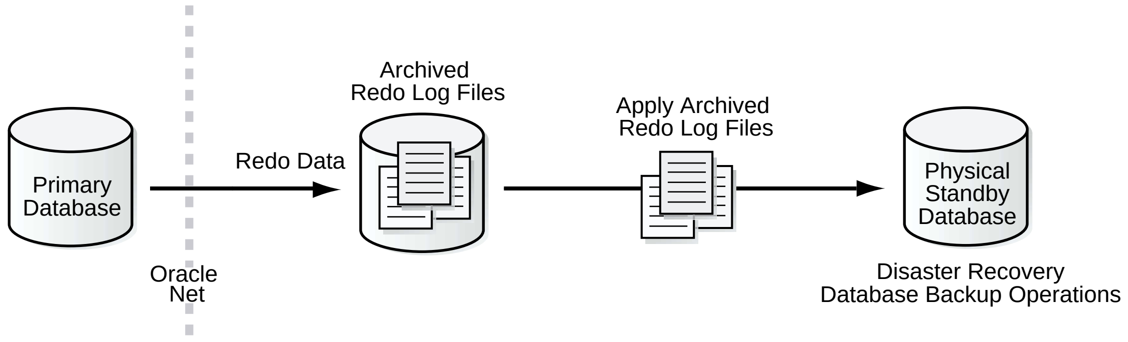
  Contains the same logical information as the production database, although the physical organization and structure of the data can be different. The logical standby database is kept synchronized with the primary database by transforming the data in the redo received from the primary database into SQL 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 instance that transmits redo data to a physical standby database. The physical standby database is remotely located from the primary database instance 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 archived redo log files are being applied to a physical standby database.

Figure 1–1 Typical Data Guard Configuration


Primary Database
Oracle Net
Redo Data
Archived Redo Log Files
Apply Archived Redo Log Files
Physical Standby Database
Disaster Recovery
Database Backup Operations

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:

- Log 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 Management Services

  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 Log Transport Services

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

Log 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

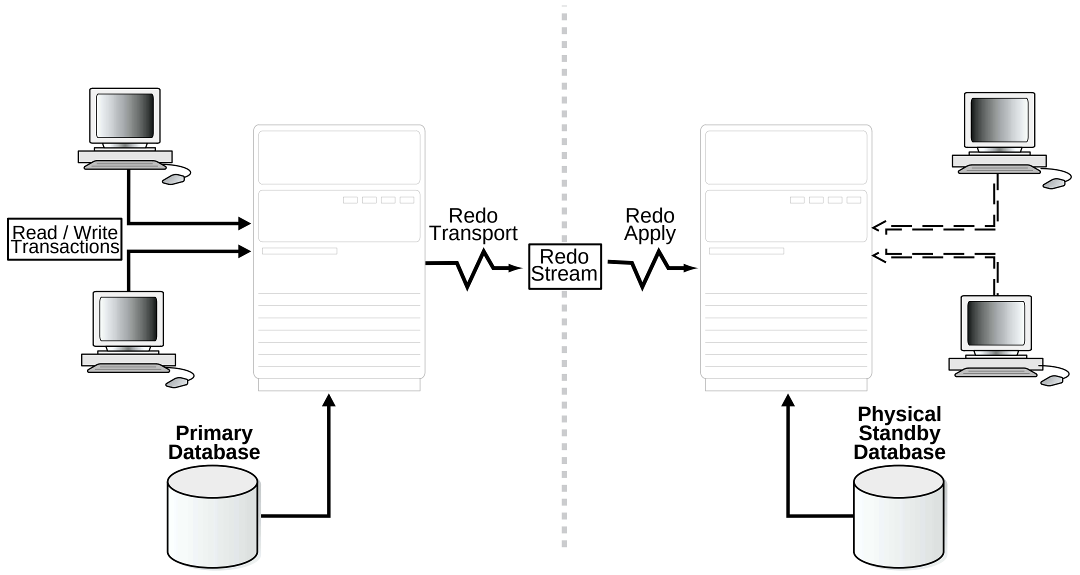
1.2.2 Log Apply Services

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 archived redo data on the standby database to maintain consistency with the primary database. It also allow 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:

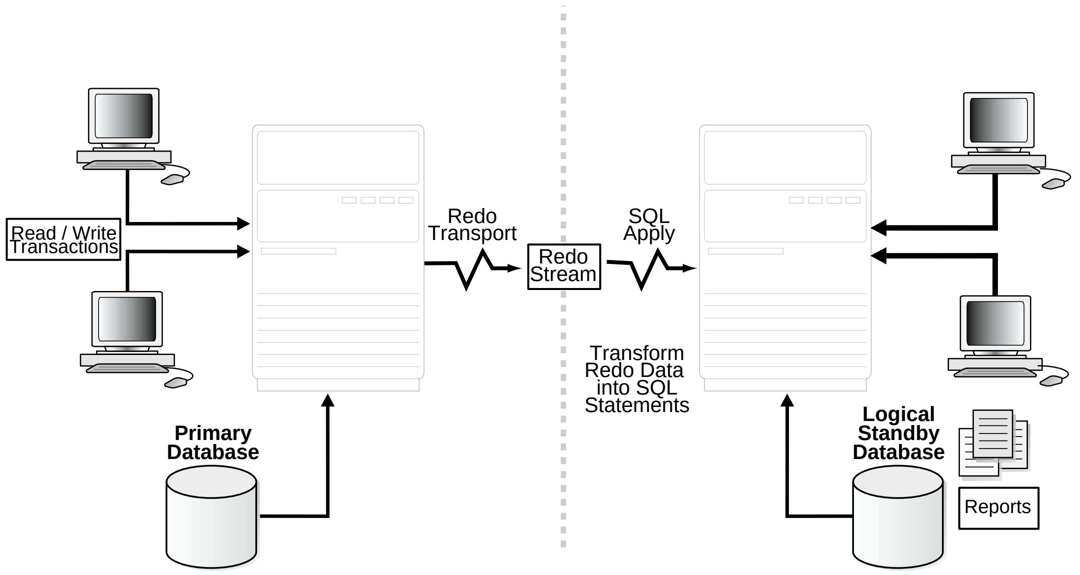
- For physical standby databases, Data Guard uses **Redo Apply** technology, which applies redo data on the standby database using standard recovery techniques of an Oracle database, as shown in Figure 1–2.

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 Management Services

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. The services that control these aspects are called **role management services.**

A **switchover** is a role reversal between the primary database and one of its standby databases. A switchover guarantees 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 an irreversible transition of a standby database to the primary role. The database administrator can configure Data Guard to ensure no data loss.

1.3 Data Guard Broker

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

- Creating and enabling Data Guard configurations, including setting up log transport services and log apply services
- Managing an entire Data Guard configuration from any system in the configuration
- Managing and monitoring Data Guard configurations that contain Real Application Clusters primary or standby databases

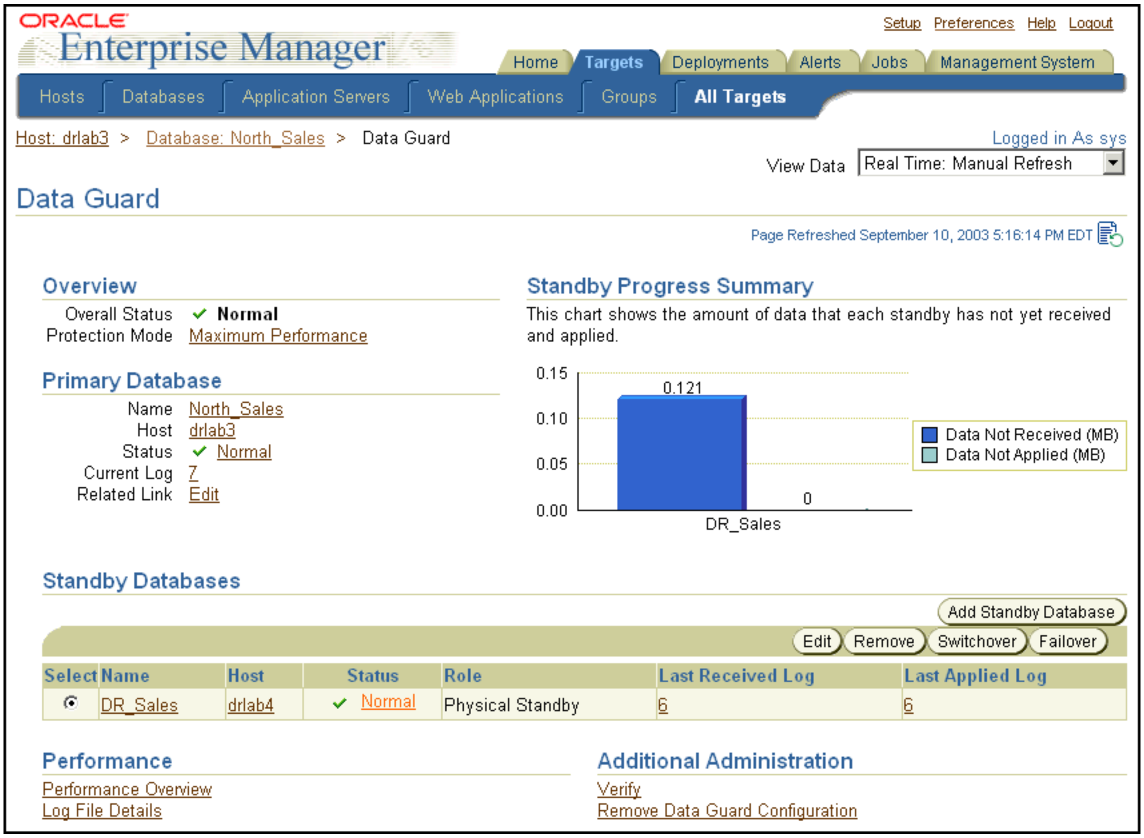
In addition, the Oracle Enterprise Manager GUI automates and simplifies:

- 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

1.3.1 Using Oracle Enterprise Manager

Oracle Enterprise Manager ("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. 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
Setup Preferences Help Logout
Home Targets Deployments Alerts Jobs Management System
Hosts Databases Application Servers Web Applications Groups All Targets
Edit Remove Switchover Failover
Add Standby Database

Host: drlab3 > Database: North_Sales > Data Guard

Logged in As sys

View Data: Real Time: Manual Refresh

Data Guard

Page Refreshed September 10, 2003 5:16:14 PM EDT

Overview

Overall Status	✓ **Normal**
Protection Mode	Maximum Performance

Primary Database

Name	North_Sales
Host	drlab3
Status	✓ Normal
Current Log	7
Related Link	Edit

Standby Progress Summary

This chart shows the amount of data that each standby has not yet received and applied.

🟦	Data Not Received (MB)
▨	Data Not Applied (MB)

Standby Databases

Select	Name	Host	Status	Role	Last Received Log	Last Applied Log
●	DR_Sales	drlab4	✓ Normal	Physical Standby	6	6

Performance

Performance Overview
Log File Details

Additional Administration

Verify
Remove Data Guard Configuration

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.

1.3.2 Using the Data Guard Command-Line Interface

The Data Guard CLI enables you to control and monitor a Data Guard configuration from the CLI prompt (DGMGRL) or within scripts. You can perform most of the activities required to manage and monitor the databases in the configuration using the CLI. See *Oracle Data Guard Broker* for complete CLI 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 a potential loss of data. The following descriptions summarize the three distinct modes of data protection.

Maximum protection This protection mode guarantees 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.

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 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 guarantees 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 databases'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 protection and maximum availability modes require that a standby redo log is 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 B, "Data Guard and Real Application Clusters" and *Oracle High Availability Architecture and Best Practices* 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 D, "Creating a Physical 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

  With standby databases, Data Guard guarantees 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.

2

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.

The following sections describe standby databases in more detail. See *Oracle High Availability Architecture and Best Practices* for information that can help you determine which type is most appropriate for your business.

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.

- **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 block for block using the data block address. The database cannot be opened while redo is being applied.

- **Open read-only**

  The 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 run a physical standby database to 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.

In either case, 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.

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 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 changes. This makes the Redo Apply technology an efficient mechanism to propagate changes among databases.

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.

- Command-line interface:
  - SQL*Plus

    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 13 for a list of the relevant statements.
  - Initialization parameters

    Several initialization parameters are used to define the Data Guard environment. See Chapter 11 for a list of the relevant initialization parameters.
- Data Guard broker command-line interface

  The Data Guard broker command-line interface is an alternative to using the Enterprise Manager GUI. The 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:

- The operating system and platform architecture on the primary and standby locations must be the same.

  For example, this means a Data Guard configuration with a primary database on a 32-bit Solaris system must have a standby database that is configured on a 32-bit Solaris system. Similarly, a primary database on a 64-bit HP-UX system must be configured with a standby database on a 64-bit HP-UX system, and a primary database on a 32-bit Linux on Intel system must be configured with a standby database on a 32-bit Linux on Intel system, and so forth.
- 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 patchset release 10.1.0.*n* to the next database 10.1.0.(*n+1*) patchset release. During a rolling upgrade, you can run different releases of the Oracle database (10.1.0.1 and higher) on the primary and logical standby databases while you upgrade them, one at a time. For complete information, see Section 9.2 and the ReadMe file for the applicable Oracle Database 10*g* patchset release.

- If you are currently running Oracle Data Guard on Oracle8*i* database software, see *Oracle Database Upgrade Guide* for complete information about upgrading to Oracle Data Guard.

- The primary database must run in ARCHIVELOG mode.
- 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 High Availability Architecture and Best Practices* 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 10.9 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 10 for a scenario describing how to set up OMF in a Data Guard configuration.

Figure 2–1 Possible Standby Configurations

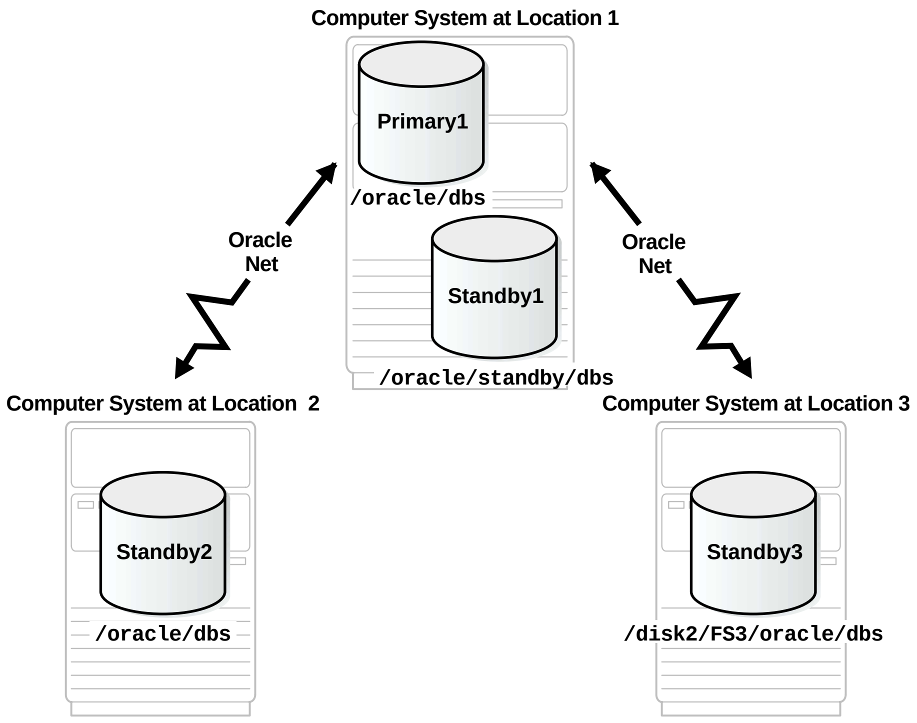


Table 2–1 describes possible configurations of primary and standby databases and the consequences of each. In the table, note that the service name defaults to the global database name, which is a concatenation of the database name (DB_NAME) and domain name (DB_DOMAIN) parameters. If you do not explicitly specify unique service names when the primary and standby databases reside on the same system, the same default global database name will be in effect for both the primary and standby databases.

Table 2–1 Standby Database Location and Directory Options

Standby System	Directory Structure	Consequences
Same as primary system	Different than primary system (required)	■ You must set the `DB_UNIQUE_NAME` initialization parameter. ■ You can either manually rename files or set up the `DB_FILE_NAME_CONVERT` and `LOG_FILE_NAME_CONVERT` 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 3.1.3.) ■ You must explicitly set up unique service names for the primary and standby databases with the `SERVICE_NAMES` initialization parameter. ■ 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.
Separate system	Same as primary system	■ 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	■ You can either manually rename files or set up the `DB_FILE_NAME_CONVERT` and `LOG_FILE_NAME_CONVERT` initialization parameters on the standby database to automatically rename the datafiles (see Section 3.1.3). ■ 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

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 associated online redo log to protect the database in case of an instance failure. Physical standby databases do not have an associated online redo log, because physical standby databases are never opened for read/write I/O; changes are not made to the database and 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 each have 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.

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 High Availability Architecture and Best Practices* 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.

2.5.2 Standby Redo Logs

A standby redo log is similar in all ways to an online redo log, except that a standby redo log is used only when the database is running in the standby role to store redo data received from the primary database.

A standby redo log is required 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 redo log destinations (described in Appendix C)

Configuring standby redo log files is highly recommended on all standby databases in a Data Guard configuration, because they provide a number of advantages:

- Because a standby redo log consists of preallocated files, a standby redo log avoids the operating system overhead of file system metadata updates common with sequential files (such as with an archive log).
- 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 5.6.2 describes how to configure standby redo log files.

3

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
- Creating a Physical Standby Database
- Further Preparations

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. Also, 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	Setting Primary Database Initialization Parameters
Section 3.1.4	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 Setting Primary Database Initialization Parameters

On the primary database, you define initialization parameters that control log 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 3–1 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 3–1 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 3–1 Primary Database: Primary Role Initialization Parameters

```
DB_NAME=chicago
DB_UNIQUE_NAME=chicago
SERVICE_NAMES=chicago
LOG_ARCHIVE_CONFIG='DG_CONFIG=(chicago,boston)'
CONTROL_FILES='/arch1/chicago/control1.ctl', '/arch2/chicago/control2.ctl'
LOG_ARCHIVE_DEST_1=
 'LOCATION=/arch1/chicago/
  VALID_FOR=(ALL_LOGFILES,ALL_ROLES)
  DB_UNIQUE_NAME=chicago'
LOG_ARCHIVE_DEST_2=
 'SERVICE=boston
  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
```

These parameters control how log transport services transmit redo data to the standby system and the archiving of redo data on the local file system. Note that the example assumes the use of the ARC*n* processes (the default) to transmit redo data. If you specify the LGWR process to transmit redo data to both the local and remote destinations, also include the `NET_TIMEOUT` attribute (described in Chapter 12) on the `LOG_ARCHIVE_DEST_2` initialization parameter.

Example 3–2 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 3–2 Primary Database: Standby Role Initialization Parameters

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

Specifying the initialization parameters shown in Example 3–2 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 Examples 3–1 and 3–2.

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.
SERVICE_NAMES	Specify a service name for this standby database that is unique from the primary database service name. If you do not explicitly specify unique service names and the primary and standby databases are located on the same system, the same default global name (consists of the database name, DB_NAME, and domain name, DB_DOMAIN, parameters) will be in effect for both databases.
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 3–1 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*	Specify where the redo data is to be archived on the primary and standby systems. In Example 3–1: ▪ 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. **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 12 for complete the LOG_ARCHIVE_DEST_*n* information.
LOG_ARCHIVE_DEST_STATE_*n*	Specify ENABLE to allow log 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.
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.
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
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.

3.1.4 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 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

Table 3–2 (Cont.) Creating a Physical Standby Database

Reference	Task	Database
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 3–3 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 3–1 and Example 3–2 are shown in bold typeface. The parameters shown in Example 3–3 are valid for the Boston database when it is running in either the primary or the standby database role.

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

```
.
.
.
DB_NAME=chicago
DB_UNIQUE_NAME=boston
SERVICE_NAMES=boston
LOG_ARCHIVE_CONFIG='DG_CONFIG=(chicago,boston)'
CONTROL_FILES='/arch1/boston/control1.ctl', '/arch2/boston/control2.ctl'
DB_FILE_NAME_CONVERT=
 '/arch1/chicago/','/arch1/boston/','/arch2/chicago/','/arch2/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
  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
INSTANCE_NAME=boston
FAL_SERVER=chicago
FAL_CLIENT=boston
.
.
.
```

Note that the example assumes the use of the ARC*n* processes (the default) to transmit redo data. If you specify the LGWR process to transmit redo data to both the local and remote destinations, also include the `NET_TIMEOUT` attribute (described in Chapter 12) 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, log 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 10*g* features, set the `COMPATIBLE` parameter to 10.1.0.0 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 3–3 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.
`SERVICE_NAMES`	Specify a service name for this standby database that is unique from the primary database service name. If you do not explicitly specify unique service names and the primary and standby databases are located on the same system, the same default global name (comprised of the database name, `DB_NAME`, and domain name, `DB_DOMAIN`, parameters) will be in effect for both databases.
`CONTROL_FILES`	Specify the path name for the control files on the standby 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
`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.
`LOG_ARCHIVE_DEST_`*n*	Specify where the redo data is to be archived. In Example 3–3: ▪ `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. **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 12 for complete information about `LOG_ARCHIVE_DEST_`*n*.
`INSTANCE_NAME`	Specify a different value for the standby database than the primary database when the primary and standby databases reside on the same system.
`FAL_SERVER`	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.
`FAL_CLIENT`	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.

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 Windows* 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 Net Services Administrator's Guide*.

Step 4 Enable broken connection detection on the standby system.

Enable broken connection detection by setting the `SQLNET.EXPIRE_TIME` parameter to 2 (minutes) in the `SQLNET.ORA` parameter file on the standby system. For example:

```
SQLNET.EXPIRE_TIME=2
```

See *Oracle Net Services Administrator's Guide*.

Step 5 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 log transport services.

The Oracle Net service name must resolve to a connect descriptor that uses the same protocol, host address, port, and SID 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 Net Services Administrator's Guide* and the *Oracle Database Administrator's Guide*.

Step 6 Create a server parameter file for the standby database.

If you plan to immediately transition the physical standby database to a logical standby database (as described in Chapter 4, "Creating a Logical Standby Database"), then skip this step and proceed with the instructions in Section 3.2.6.

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 statements to start and mount the database in read-only mode:

```
SQL> STARTUP OPEN READ ONLY;
```

Do not open the database; it should remain closed to user access; a physical standby database must be in the mounted state (or open in read-only mode) to receive redo data.

Step 2 Create a new temporary file for the physical standby database.

If you plan to immediately transition the physical standby database to a logical standby database (as described in Chapter 4, "Creating a Logical Standby Database"), then skip this step and proceed with the instructions in Step 3.

Creating a new temporary file on the physical standby database now, rather than later, is beneficial. Temporary files enable disk sorting when the database is open in read-only mode and prepare the database for future role transitions.

To add temporary files to the physical standby database, perform the following tasks:

1. Identify the tablespaces that should contain temporary files. Do this by entering the following command on the standby database:

   ```
   SQL> SELECT TABLESPACE_NAME FROM DBA_TABLESPACES
     2>  WHERE CONTENTS = 'TEMPORARY';

   TABLESPACE_NAME
   --------------------------------
   TEMP1
   TEMP2
   ```

2. Add new temporary files to the standby database.

   For each tablespace identified in the previous query, add a new temporary file to the standby database. The following example adds a new temporary file called `TEMP1` with size and reuse characteristics that match the primary database temporary files:

   ```
   SQL> ALTER TABLESPACE TEMP1 ADD TEMPFILE
   ```

```
2> '/arch1/boston/temp01.dbf'
3> SIZE 40M REUSE;
```

Note: To create temporary files on the physical standby database that match the temporary files on the primary database, query the V$TEMPFILE view on the primary database to obtain complete information about the primary database temporary files.

Step 3 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;
```

This statement automatically mounts the database. Also, 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 4 Test archival operations to the physical standby database.

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 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 log 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 ARCHIVE LOG CURRENT` statement to force a log switch and archive the current online redo log file group:

```
SQL> ALTER SYSTEM ARCHIVE LOG CURRENT;
```

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 6.3.4, "Monitoring Log Apply Services on Physical Standby Databases" to verify log transport services and log apply services are working correctly.

3.3 Further Preparations

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.

- Configure standby redo logs

  Standby redo logs are required for standby databases running in the maximum protection mode and maximum availability mode. However, configuring standby redo logs is recommended on all standby databases, because during a failover Data Guard can recover and apply more redo data from standby redo log files than from the archived redo log files alone. The standby redo logs should exist on both primary and standby databases and have the same size and names. See Section 5.6.2 for more information.

- Enable Flashback Database

  Flashback Database removes the need to re-create the primary database after a failover. Flashback Database is similar to conventional point-in-time recovery in its effects, enabling you to return a database to its state at a time in the recent past. Flashback Database is faster than point-in-time recovery, because it does not require restoring datafiles from backup or the extensive application of redo data. You can enable Flashback Database on the primary database, the standby database, or both. See *Oracle Database Backup and Recovery Advanced User's Guide* for more information.

4 Creating a Logical Standby Database

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

- Preparing for Logical Standby Database Creation
- Creating a Logical Standby Database
- Further Preparations

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.

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 Preparing for Logical Standby Database Creation

Before you create a 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 Datatypes 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 Datatypes and Storage Attributes for Tables

Before setting up a logical standby database, ensure the logical standby database can maintain the datatypes and tables in your primary database.

The following list shows the various database objects that are supported and unsupported in logical standby databases.

Supported Datatypes and Storage Attributes for Tables

`CHAR`
`NCHAR`
`VARCHAR2` and `VARCHAR`
`NVARCHAR2`
`NUMBER`
`DATE`
`TIMESTAMP`
`TIMESTAMP WITH TIME ZONE`
`TIMESTAMP WITH LOCAL TIME ZONE`
`INTERVAL YEAR TO MONTH`
`INTERVAL DAY TO SECOND`
`RAW`
`CLOB` (including both fixed-width and variable-width character sets)
`NCLOB`
`BLOB`
`LONG`
`LONG RAW`
`BINARY_FLOAT`
`BINARY_DOUBLE`
Index-organized tables (without overflows and without `LOB` columns)

Unsupported Datatypes

`BFILE`
`ROWID`
`UROWID`
user-defined types

object types REFs
varrays
nested tables
XMLType

Unsupported Tables, Sequences, and Views

- Most schemas that ship with the Oracle database are skipped by SQL Apply
- Tables with unsupported datatypes
- Tables using table compression

To determine exactly which schemas will be skipped, query the DBA_LOGSTDBY_SKIP view.

To determine if the primary database contains unsupported objects, query the DBA_LOGSTDBY_UNSUPPORTED view. See Chapter 14, "Views Relevant to Oracle Data Guard" for more information about the DBA_LOGSTDBY_UNSUPPORTED view.

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 datatypes, table, sequences, or views on the primary database will not be propagated to the logical standby database. Moreover, no error message will be returned.

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 datatypes 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, log apply services automatically exclude 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.

4.1.1.1 Skipped SQL Statements on a Logical Standby Database

By default, all SQL statements except those in the following list are applied to a logical standby database if they are executed on a primary database:

```
ALTER DATABASE
ALTER SESSION
ALTER MATERIALIZED VIEW
ALTER MATERIALIZED VIEW LOG
ALTER SYSTEM
CREATE CONTROL FILE
CREATE DATABASE
CREATE DATABASE LINK
CREATE PFILE FROM SPFILE
CREATE SCHEMA AUTHORIZATION
CREATE MATERIALIZED VIEW
CREATE MATERIALIZED VIEW LOG
CREATE SPFILE FROM PFILE
DROP DATABASE LINK
DROP MATERIALIZED VIEW
DROP MATERIALIZED VIEW LOG
EXPLAIN
LOCK TABLE
SET CONSTRAINTS
SET ROLE
SET TRANSACTION
```

4.1.1.2 Supported Objects and Operations

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 include DBMS_OUTPUT, DBMS_RANDOM, DBMS_PIPE, DBMS_DESCRIBE, DBMS_OBFUSCATION_TOOLKIT, DBMS_TRACE, DBMS_METADATA, and so on.

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 user data is in the category of supported datatypes. Examples of such packages include DBMS_LOB, DBMS_SQL, DBMS_TRANSACTION, and so on.

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 include DBMS_JAVA, DBMS_REGISTRY, DBMS_ALERT, DBMS_SPACE_ADMIN, DBMS_REFRESH, DBMS_SCHEDULER, DBMS_AQ, and so on.

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.

See *PL/SQL Packages and Types Reference* for more information about all of the Oracle PL/SQL supplied packages.

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

Because the ROWIDs on a logical standby database might not be the same as the ROWIDs on the primary database, a different mechanism must be used to match the updated row on the primary database to its corresponding row on the logical standby database. You can use one of the following to match up the corresponding rows:

- Primary key
- Unique index

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

Perform the following steps to ensure SQL Apply can uniquely identify table rows.

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

Query the DBA_LOGSTDBY_NOT_UNIQUE view to identify tables in the primary database that do not have a primary key or unique index with NOT NULL columns. The following query displays a list of tables that SQL Apply might not be able to uniquely identify:

```
SQL> SELECT OWNER, TABLE_NAME,BAD_COLUMN FROM DBA_LOGSTDBY_NOT_UNIQUE
  2> WHERE TABLE_NAME NOT IN (SELECT TABLE_NAME FROM DBA_LOGSTDBY_UNSUPPORTED);
```

Some of the tables displayed in the DBA_LOGSTDBY_NOT_UNIQUE view can still be supported because **supplemental logging** (that you will enable in Section 4.2.2.1) adds information that uniquely identifies the row containing the redo data. The presence or absence of a primary key or unique index can affect supplemental logging as follows:

- If the table has a primary key or a unique index with NOT NULL columns, the amount of information added to the online redo log file during supplemental logging is minimal.
- If the table does not have a primary key or a unique index, supplemental logging automatically logs all scalar values for each row to the online redo log file.

The value of the BAD_COLUMN column will be either Y or N, as described in the following list:

- Y

  Indicates a table column is defined using an unbounded datatype, such as CLOB or BLOB. SQL Apply attempts to maintain these tables, but you must ensure the application provides uniqueness in bounded columns only. Note that if two rows in the table match except for rows in the LOB column, then the table cannot be maintained properly and SQL Apply will stop.

- N

  Indicates the table contains enough column information to maintain the table in a logical standby database.

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. See *Oracle Database SQL Reference* for ALTER TABLE statement syntax and usage information.

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 where rows can be uniquely identified using the id and name columns:

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

The RELY constraint tells the system to assume the rows are unique. Be careful to select columns for the disabled RELY constraint that will uniquely identify a row. If the columns selected for the RELY constraint do not uniquely identify the row, SQL Apply fails to apply data from the archived redo log file or standby redo log file to the logical standby database.

To improve the performance of SQL Apply, add an index to the columns that uniquely identify the row on the logical standby database. Failure to do this results in full table scans.

See *Oracle Database Reference* for more information about the DBA_LOGSTDBY_NOT_UNIQUE view, *Oracle Database SQL Reference* for more information about creating RELY constraints, and Section 9.4 for information about RELY constraints and actions you can take to increase performance on a logical standby database.

4.2 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 or databases 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	Prepare the Primary Database to Support a Logical Standby Database	Primary
Section 4.2.3	Prepare to Transition to a Logical Standby Database	Standby
Section 4.2.4	Start the Logical Standby Database	Standby
Section 4.2.5	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 into a logical standby database, as follows:

Step 1 Create a physical standby database.

Follow the instructions in Chapter 3 to create a physical standby database.

Step 2 Ensure the physical standby database is caught up to the primary database.

After you complete the steps in Section 3.2.6 to start the physical standby database and Redo Apply, allow recovery to continue until the physical standby database is consistent with the primary database, including all database structural changes (such as adding or dropping datafiles).

4.2.2 Prepare the Primary Database to Support a Logical Standby Database

This section contains the following topics:

- Ensure Supplemental Logging Is Enabled
- Prepare the Primary Database for Role Transitions

4.2.2.1 Ensure Supplemental Logging Is Enabled

Supplemental logging must be enabled on the primary database to support a logical standby database. Because an Oracle Database only logs the columns that were modified, this is not always sufficient to uniquely identify the row that changed and additional (supplemental) information must be put into the stream of redo data. The supplemental information that is added to the redo data helps SQL Apply to correctly identify and maintain tables in the logical standby database.

Step 1 Determine if supplemental logging is enabled.

To determine if supplemental logging is enabled on the primary database, query the V$DATABASE fixed view. For example:

```
SQL> SELECT SUPPLEMENTAL_LOG_DATA_PK AS PK_LOG,
  2> SUPPLEMENTAL_LOG_DATA_UI AS UI_LOG
  3> FROM V$DATABASE;

PK_LOG     UI_LOG
------     ------
    NO         NO
```

In this example, the NO values indicate that supplemental logging is *not* enabled on the primary database.

If supplemental logging is enabled, then go to Section 4.2.2.2. If supplemental logging is not enabled, then perform the following steps to enable supplemental logging.

Step 2 Enable supplemental logging.

On the primary database, issue the following statement to add primary key and unique index information to the archived redo log file:

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

This SQL statement adds the information to uniquely identify the row that changed on the primary database so SQL Apply can correctly identify and maintain the same row on the standby database.

This statement may take a long time to finish on an open database because it waits for all ongoing transactions to finish. All redo that is generated after the completion of this statement is guaranteed to have supplemental logging information.

Step 3 Verify supplemental logging is enabled.

On the primary database, verify supplemental logging is enabled by issuing the same query used previously. For example:

```
SQL> SELECT SUPPLEMENTAL_LOG_DATA_PK AS PK_LOG,
  2>  SUPPLEMENTAL_LOG_DATA_UI AS UI_LOG
  3>  FROM V$DATABASE;

PK_LOG UI_LOG
------ ------
   YES    YES
```

In this example, the YES values indicate supplemental logging is enabled on the primary database. For all tables with a primary key (SUPPLEMENTAL_LOG_DATA_PK) or unique index (SUPPLEMENTAL_LOG_DATA_UI), all columns of the primary key and unique index are placed into the online redo log file whenever an update operation is performed.

Note: If you enable supplemental logging on a primary database in a Data Guard configuration that also contains physical standby databases, then you must issue the `ALTER DATABASE ADD SUPPLEMENTAL LOG DATA` statement on each physical standby database to ensure future switchovers work correctly.

See Chapter 14, "Views Relevant to Oracle Data Guard" for more information about the `V$DATABASE` view and the *Oracle Database SQL Reference* for more information about the `ALTER DATABASE ADD SUPPLEMENTAL LOG DATA` statements.

4.2.2.2 Prepare the Primary Database for Role Transitions

In Section 3.1.3, 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, you must modify the parameters on the primary database, as shown in Example 4–1, so that no parameters need to change after a role transition.

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

```
LOG_ARCHIVE_DEST_1=
 'LOCATION=/arch1/chicago/
  VALID_FOR=(ONLINE_LOGFILES,ALL_ROLES)
  DB_UNIQUE_NAME=chicago'
LOG_ARCHIVE_DEST_2=
 'SERVICE=boston
  VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)
  DB_UNIQUE_NAME=boston'
LOG_ARCHIVE_DEST_3=
 'LOCATION=/arch2/chicago/
  VALID_FOR=(STANDBY_LOGFILES,STANDBY_ROLE)
  DB_UNIQUE_NAME=chicago'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_STATE_2=ENABLE
LOG_ARCHIVE_DEST_STATE_3=ENABLE
UNDO_RETENTION=3600
```

To dynamically set the `LOG_ARCHIVE_DEST_`*n* parameters, 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. Also, set the `UNDO_RETENTION` parameter to 3600; this parameter specifies (in seconds) the amount of committed undo information to retain in the database.

Setting the value to 3600 is recommended for best results when building a LogMiner dictionary for the logical standby database.

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_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/`.	Archives 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/`.
`LOG_ARCHIVE_DEST_2`	Transmits the redo data to the remote logical standby database `boston`.	Is ignored; `LOG_ARCHIVE_DEST_2` is valid only when `chicago` is running in the primary role.
`LOG_ARCHIVE_DEST_3`	Is ignored; `LOG_ARCHIVE_DEST_3` is valid only when `chicago` is running in the standby role.	Archives redo data received from the primary database to the local archived redo log files in `/arch2/chicago/`.

4.2.3 Prepare to 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:

- Ensure Supplemental Logging Is Enabled
- Prepare an Initialization Parameter File for the Logical Standby Database
- Create a Control File for the Logical Standby Database

4.2.3.1 Ensure Supplemental Logging Is Enabled

Enabling supplemental logging on the logical standby database now rather than later is beneficial to prepare the database for future role transitions. Use the steps described in Section 4.2.2.1, but perform them on the logical standby database instead of on the primary database.

4.2.3.2 Prepare an Initialization Parameter File for the Logical Standby Database

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

Step 1 Set initialization parameters for the logical standby database.

In the text initialization parameter file (PFILE) that you created in Section 3.2.3, you need to make some additional modifications to the `LOG_ARCHIVE_DEST_`*n* parameters and add the `PARALLEL_MAX_SERVERS` parameter.

You need to modify the LOG_ARCHIVE_DEST_*n* 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
   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
PARALLEL_MAX_SERVERS=9
UNDO_RETENTION=3600
```

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/`.

In Example 4–2, the `PARALLEL_MAX_SERVERS` initialization parameter was added to the parameter file to specify the maximum number of parallel servers working on the logical standby database. This parameter is required for logical standby databases. Do not set `PARALLEL_MAX_SERVERS` to a value less than 5; for best results, set it to a minimum of 9. See Section 9.4 for more details.

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. Use the `SHOW PARAMETERS` command to verify no other initialization parameters need to be changed.

Step 2 Shut down the logical standby database.

To shut down the logical standby database, issue the following:

```
SQL> SHUTDOWN IMMEDIATE;
```

You will mount the logical standby database using the new initialization parameter file later, in Section 4.2.4.

4.2.3.3 Create a Control File for the Logical Standby Database

Perform the following steps to create a control file for the standby database.

Step 1 Create the logical standby control file.

Issue the `ALTER DATABASE CREATE LOGICAL STANDBY CONTROLFILE` statement to create a control file for the standby database. You must include the `LOGICAL` keyword when creating a logical standby database, as shown in the following example:

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

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

Step 2 Copy the control file to the logical standby system.

On the primary system, use an operating system copy utility to copy the standby control file to the logical standby system. For example, the following examples use the UNIX `cp` command:

```
cp /tmp/boston.ctl /arch1/boston/control1.ctl
cp /tmp/boston.ctl /arch2/boston/control2.ctl
```

4.2.4 Start the Logical Standby Database

Perform the following steps to start, mount, and activate the logical standby database and SQL Apply.

Step 1 Start and mount the logical standby database.

On the logical standby database, 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> STARTUP MOUNT PFILE=initboston.ora;
```

Step 2 Prepare for SQL Apply.

On the logical standby database, issue the following statement to prepare it for SQL Apply:

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

Step 3 Activate the logical standby database.

Issue the following statement to activate this database as a logical standby database:

```
SQL> ALTER DATABASE ACTIVATE STANDBY DATABASE;
```

Step 4 Reset the database name of the logical standby database.

Run the Oracle DBNEWID (nid) utility to change the database name of the logical standby database. Changing the name prevents any interaction between this copy of the primary database and the original primary database.

Before you run the DBNEWID (nid) utility, you must shut down and mount the database:

```
SQL> SHUTDOWN IMMEDIATE;
SQL> STARTUP MOUNT PFILE=initboston.ora;
```

Now, run the Oracle DBNEWID utility on the logical standby database to change the database name and shut it down:

```
nid TARGET=SYS/password@boston DBNAME=boston
Connected to database chicago (DBID=1456557175)

Control Files in database:
    /arch1/boston/control1.ctl
Change database ID and database name chicago to boston? (Y/[N]) => y

Proceeding with operation
Changing database ID from 1456557175 to 416458362
Changing database name from chicago to boston
    Control File /arch1/boston/control1.ctl - modified
    Datafile /arch1/boston/system01.dbf - dbid changed, wrote new name
    Datafile /arch1/boston/undotbs01.dbf -dbid changed, wrote new name
    .
    .
    .
    Control File /arch1/boston/control1.ctl-dbid changed, wrote new name

Database name changed to boston.
Modify parameter file and generate a new password file before restarting.
Database ID for database boston change to 416458362.
All previous backups and archive logs for this database are unusable.
Database has been shut down, open database with RESETLOGS option.
Successfully changed database name and ID.
DBNEWID - Completed successfully.
```

You must re-create the password file after running the Oracle DBNEWID (nid) utility.

Step 5 Change the logical standby database name in the parameter file.

The output from the DBNEWID utility states that you must update the initialization parameter file. The following steps describe how to perform this task.

1. Modify the `DB_NAME` initialization parameter.

   Set the `DB_NAME` initialization parameter in the text initialization parameter file from Section 4.2.3.2 to match the new name:

   ```
   .
   .
   .
   DB_NAME=boston
   .
   .
   .
   ```

2. Create a server parameter file for the logical standby database.

   Connect to an idle instance of the logical standby database, and create a server parameter file for the standby database from the text initialization parameter file. For example:

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

3. Restart the logical standby database.

   Start and open the logical standby database to user access, as follows:

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

Step 6 Change the logical standby database global name.

Each database should have a unique global name. To change the global name of the standby database to boston, issue the following statement:

```
SQL> ALTER DATABASE RENAME GLOBAL_NAME TO boston;
```

Step 7 Create a new temporary file for the logical standby database.

Creating a new temporary file on the logical standby database now, rather than later, is beneficial to prepare the database for future role transitions.

To add temporary files to the logical standby database, perform the following tasks:

1. Identify the tablespaces that should contain temporary files. Do this by entering the following statement on the standby database:

```
SQL> SELECT TABLESPACE_NAME FROM DBA_TABLESPACES
  2>  WHERE CONTENTS = 'TEMPORARY';

TABLESPACE_NAME
--------------------------------
TEMP1
TEMP2
```

2. Add new temporary files to the standby database.

   For each tablespace identified in the previous query, add a new temporary file to the standby database. The following example adds a new temporary file called `TEMP1` with size and reuse characteristics that match the primary database temporary files:

   ```
   SQL> ALTER TABLESPACE TEMP1 ADD TEMPFILE
     2> '/arch1/boston/temp01.dbf'
     3> SIZE 40M REUSE;
   ```

   Note: To create temporary files on the logical standby database that match the temporary files on the primary database, query the `V$TEMPFILE` view on the primary database to obtain complete information about the primary database temporary files.

Step 8 Start SQL Apply.

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

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

Continue with Section 4.2.5 to verify the logical standby database is performing properly. See Section 5.6.2 to configure standby redo log files, and see Section 6.4 for information about SQL Apply and real-time apply.

4.2.5 Verify the Logical Standby Database Is Performing Properly

Once you create a logical standby database and set up log transport services and log apply services, verify redo data is being transmitted from the primary database and applied to the standby database. To check this, perform the following steps.

Step 1 Verify the archived redo log files were registered.

To verify the archived redo log files were registered on the logical standby system, connect to the logical standby database and query the DBA_LOGSTDBY_LOG view. For example:

```
SQL> ALTER SESSION SET NLS_DATE_FORMAT  = 'DD-MON-YY HH24:MI:SS';
Session altered.

SQL> COLUMN DICT_BEGIN FORMAT A10
SQL> COLUMN DICT_END FORMAT A8
SQL> SELECT SEQUENCE#, FIRST_TIME, NEXT_TIME, DICT_BEGIN, DICT_END
  2> FROM DBA_LOGSTDBY_LOG ORDER BY SEQUENCE#;

 SEQUENCE# FIRST_TIME         NEXT_TIME          DIC DIC
---------- ------------------ ------------------ --- ---
        24 23-JUL-02 18:19:05 23-JUL-02 18:19:48 YES YES
        25 23-JUL-02 18:19:48 23-JUL-02 18:19:51 NO  NO
        26 23-JUL-02 18:19:51 23-JUL-02 18:19:54 NO  NO
        27 23-JUL-02 18:19:54 23-JUL-02 18:19:59 NO  NO
        28 23-JUL-02 18:19:59 23-JUL-02 18:20:03 NO  NO
        29 23-JUL-02 18:20:03 23-JUL-02 18:20:13 NO  NO
        30 23-JUL-02 18:20:13 23-JUL-02 18:20:18 NO  NO
        31 23-JUL-02 18:20:18 23-JUL-02 18:20:21 NO  NO

8 rows selected.
```

Step 2 Send redo data to the standby database.

Connect to the primary database and enter the following statement to begin sending redo data to the standby database:

```
SQL> ALTER SYSTEM ARCHIVE LOG CURRENT;
System altered.

SQL> ALTER SYSTEM ARCHIVE LOG CURRENT;
System altered.
```

Step 3 Query the DBA_LOGSTDBY_LOG view again.

Connect to the logical standby database and query the DBA_LOGSTDBY_LOG view again:

```
SQL> ALTER SESSION SET NLS_DATE_FORMAT  = 'DD-MON-YY HH24:MI:SS';
Session altered.

SQL> COLUMN DICT_BEGIN FORMAT A10
SQL> COLUMN DICT_END FORMAT A8
SQL> SELECT SEQUENCE#, FIRST_TIME, NEXT_TIME, DICT_BEGIN, DICT_END
  2  FROM DBA_LOGSTDBY_LOG ORDER BY SEQUENCE#;

 SEQUENCE# FIRST_TIME         NEXT_TIME          DIC DIC
---------- ------------------ ------------------ --- ---
        24 23-JUL-02 18:19:05 23-JUL-02 18:19:48 YES YES
        25 23-JUL-02 18:19:48 23-JUL-02 18:19:51 NO  NO
        26 23-JUL-02 18:19:51 23-JUL-02 18:19:54 NO  NO
        27 23-JUL-02 18:19:54 23-JUL-02 18:19:59 NO  NO
        28 23-JUL-02 18:19:59 23-JUL-02 18:20:03 NO  NO
        29 23-JUL-02 18:20:03 23-JUL-02 18:20:13 NO  NO
        30 23-JUL-02 18:20:13 23-JUL-02 18:20:18 NO  NO
        31 23-JUL-02 18:20:18 23-JUL-02 18:20:21 NO  NO
        32 23-JUL-02 18:20:21 23-JUL-02 18:32:11 NO  NO
        33 23-JUL-02 18:32:11 23-JUL-02 18:32:19 NO  NO

10 rows selected.
```

By checking the files on the standby database, archiving a few log files, and then checking the standby database again, you can see that the new archived redo log files were registered. These log files are now available for log apply services to begin applying them.

Step 4 Verify redo data is being applied correctly.

On the logical standby database, query the V$LOGSTDBY_STATS view to verify redo data is being applied correctly. For example:

```
SQL> COLUMN NAME FORMAT A30
SQL> COLUMN VALUE FORMAT A30
SQL> SELECT NAME, VALUE FROM V$LOGSTDBY_STATS WHERE NAME = 'coordinator state';

NAME                           VALUE
------------------------------ ------------------------------
coordinator state              INITIALIZING
```

In the example, the output from the V$LOGSTDBY_STATS view shows the coordinator process is in the initialization state. When the coordinator process is initializing, log apply services are preparing to begin SQL Apply, but data from the archived redo log files is not being applied to the logical standby database.

Knowing the state of the coordinator process is of particular importance because it is the LSP background process that instructs all of the other logical standby processes. Section 9.1.9 describes the LSP background processes in more detail.

Step 5 View the V$LOGSTDBY view to see current SQL Apply activity.

On the logical standby database, query the V$LOGSTDBY view to see a current snapshot of SQL Apply activity. A text message describing the current activity of each process involved in reading and applying changes is displayed.

Example 4–3 shows typical output during the initialization phase.

Example 4–3 V$LOGSTDBY Output During the Initialization Phase

```
SQL> COLUMN STATUS FORMAT A50

SQL> COLUMN TYPE FORMAT A12

SQL> SELECT TYPE, HIGH_SCN, STATUS FROM V$LOGSTDBY;
TYPE           HIGH_SCN STATUS
------------ ---------- --------------------------------------------------
COORDINATOR             ORA-16115: loading Log Miner dictionary data
READER                  ORA-16127: stalled waiting for additional transact
                        ions to be applied
BUILDER                 ORA-16117: processing
PREPARER                ORA-16116: no work available

SQL> SELECT TYPE, HIGH_SCN, STATUS FROM V$LOGSTDBY;
TYPE           HIGH_SCN STATUS
------------ ---------- --------------------------------------------------
COORDINATOR             ORA-16126: loading table or sequence object number
READER                  ORA-16116: no work available
BUILDER                 ORA-16116: no work available
PREPARER                ORA-16116: no work available
```

Once the coordinator process begins applying redo data to the logical standby database, the V$LOGSTDBY view indicates this by showing the APPLYING state.

Example 4–4 shows typical output during the applying phase. Notice that the values in the HIGH_SCN column continue to increment. The numbers in this column will continue to increase as long as changes are being applied. The HIGH_SCN column serves only as an indicator of progress.

Example 4–4 V$LOGSTDBY Output During the Applying Phase

```
SQL> COLUMN NAME FORMAT A30
SQL> COLUMN VALUE FORMAT A30
SQL> SELECT NAME, VALUE FROM V$LOGSTDBY_STATS WHERE NAME = 'coordinator state';
NAME                           VALUE
------------------------------ ------------------------------
```

```
coordinator state              APPLYING

SQL> COLUMN STATUS FORMAT A50
SQL> COLUMN TYPE FORMAT A12
SQL> SELECT TYPE, HIGH_SCN, STATUS FROM V$LOGSTDBY;
TYPE           HIGH_SCN STATUS
------------ ---------- --------------------------------------------------
COORDINATOR             ORA-16117: processing
READER                  ORA-16127: stalled waiting for additional transact
                        ions to be applied

BUILDER          191896 ORA-16116: no work available
PREPARER         191902 ORA-16117: processing
ANALYZER         191820 ORA-16120: dependencies being computed for transac
                        tion at SCN 0x0000.0002ed4e

APPLIER          191209 ORA-16124: transaction 1 16 1598 is waiting on ano
                        ther transaction

APPLIER          191205 ORA-16116: no work available
APPLIER          191206 ORA-16124: transaction 1 5 1603 is waiting on anot
                        her transaction

APPLIER          191213 ORA-16117: processing
APPLIER          191212 ORA-16124: transaction 1 20 1601 is waiting on ano
                        ther transaction

APPLIER          191216 ORA-16124: transaction 1 4 1602 is waiting on anot
                        her transaction

11 rows selected.
```

Step 6 Check the overall progress of SQL Apply.

To check the overall progress of SQL Apply, query the DBA_LOGSTDBY_PROGRESS view on the standby database. For example:

```
SQL> SELECT APPLIED_SCN, NEWEST_SCN FROM DBA_LOGSTDBY_PROGRESS;

APPLIED_SCN NEWEST_SCN
----------- ----------
     180702     180702
```

If standby redo log files are not configured, the numbers in the APPLIED_SCN and NEWEST_SCN columns are equal (as shown in the query example), indicating that all of the available data in the archived redo log file was applied. These values can be compared to the values in the FIRST_CHANGE# column in the DBA_LOGSTDBY_LOG view to see how much log file information has to be applied and how much remains. If standby redo log files are configured, the numbers in the APPLIED_SCN and NEWEST_SCN columns may be close, but they will seldom be equal.

See Section 5.9.1, "Monitoring Log File Archival Information" and Section 6.4.4, "Monitoring Log Apply Services for Logical Standby Databases" for information about how to verify both log transport and log apply services are working correctly.

4.3 Further Preparations

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 for information about the data protection modes and how to upgrade or downgrade the current protection mode.

- Configure standby redo logs

  Standby redo logs are required for standby databases running in the maximum protection mode and maximum availability mode. However, configuring standby redo logs is recommended on all standby databases, because during a failover Data Guard can recover and apply more redo data from standby redo log files than from the archived redo log files alone. The standby redo logs should exist on both primary and standby databases and have the same size and names. See Section 5.6.2, "Configuring Standby Redo Log Files" for more information.

- Enable Flashback Database

  Flashback Database removes the need to re-create the primary database after a failover. Flashback Database is similar to conventional point-in-time recovery in its effects, enabling you to return a database to its state at a time in the recent past. Flashback database is faster than point-in-time recovery, because it does not require restoring datafiles from backup or the extensive application of redo data. You can enable Flashback Database on the primary database, the standby database, or both. See *Oracle Database Backup and Recovery Advanced User's Guide* for more information.

5 Log Transport Services

This chapter describes configuring log transport services to transmit redo from the production database to one or more archival destinations. It contains the following topics:

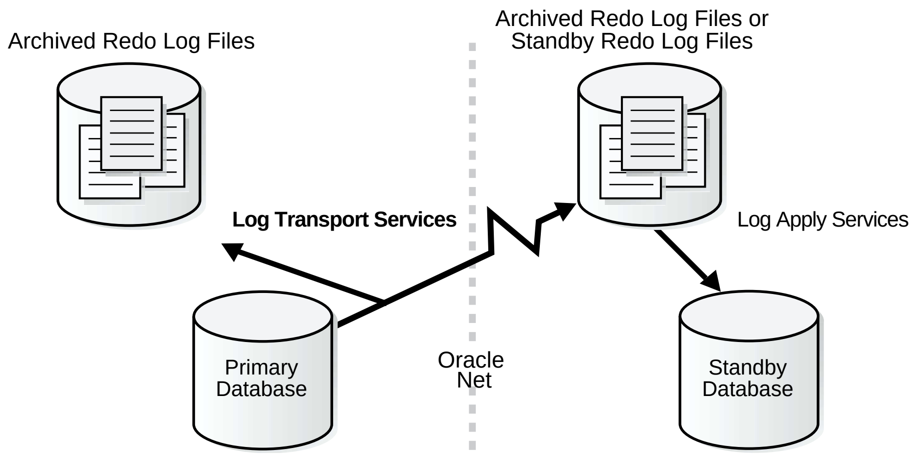
5.1 Introduction to Log Transport Services

Log transport services control the automated transfer of redo data from a production or primary database destination to another (standby) database destination. Log transport services also manage the process of resolving any gaps in the archived redo log files due to a network failure.

Log 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, cross-instance archival database environments, Oracle Change Data Capture staging databases, and Oracle Streams downstream capture databases.

Figure 5–1 shows a simple Data Guard configuration with log 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 at 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
- Configuring Destinations with the LOG_ARCHIVE_DEST_n Parameter
- Setting Up Flash Recovery Areas As Destinations

5.2.1 Destination Types

There are several types of destinations supported by log 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.

- Cross-instance archival database environment

  A cross-instance archival database environment is possible on the primary and standby databases. Within a Real Application Clusters environment, each instance transmits its redo data to a single instance of the cluster. This instance, known as the recovery instance, is typically the instance where recovery is performed. The recovery instance typically has a tape drive available for RMAN backup and restoration support. Cross-instance archival environments are described in Appendix B. *Oracle High Availability Architecture and Best Practices* provides additional information about RAC and Data Guard configurations.

- Oracle Streams downstream capture database

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

- Oracle Change Data Capture staging database

  This destination type allows Change Data Capture for the Asynchronous AutoLog to use log transport services to transfer redo data from a source database to a remote staging database where a process captures change data from the archived redo log files. See *Oracle 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, ... 10) destinations, each of which must specify either the LOCATION or the SERVICE attribute to specify where to archive the redo data. (Also, see Chapter 12 for complete information about all LOG_ARCHIVE_DEST_*n* attributes.)

The LOCATION and SERVICE attributes describe either a local disk location or an Oracle Net service name that represents a standby destination to which log transport services will transmit redo data. 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 every LOG_ARCHIVE_DEST_*n* initialization parameter that you define, you can 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	Log transport services can transmit redo data to this destination. ENABLE is the default.
DEFER	Log 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.

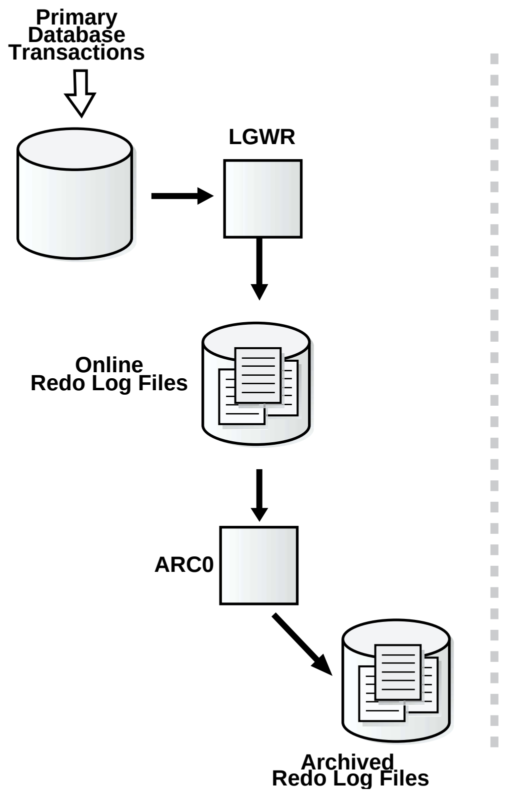
Example 5–1 provides an example of one destination with the LOCATION attribute.

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 basic Data Guard configuration that provides disaster recovery, you need to add a standby database at a remote destination by specifying the `SERVICE` attribute.

Example 5–2 shows the initialization parameters that enable log 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 basic Data Guard configuration that is based on the premises that log transport services will use archiver (ARC*n*) processes to archive to both the local and remote destinations, and the configuration provides the maximum performance level of data protection.

Although you can create a basic 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. The following sections describe several of the `LOG_ARCHIVE_DEST_`*n* parameter attributes.

5.2.3 Setting Up Flash Recovery Areas As Destinations

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

To configure a flash recovery area, you specify the directory, file system, 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, Data Guard implicitly uses the `LOG_ARCHIVE_DEST_10` destination to refer to the default disk location for the flash recovery area and for storing the archived redo log files. (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.

Although the flash recovery area uses the `LOG_ARCHIVE_DEST_10` destination by default, you can explicitly set up flash recovery areas to use one or more other `LOG_ARCHIVE_DEST_`*n* destinations or the `STANDBY_ARCHIVE_DEST` destination. This section contains the following topics:

- Using the LOG_ARCHIVE_DEST_10 Default Flash Recovery Area
- Setting Flash Recovery Areas to Other LOG_ARCHIVE_DEST_n Destinations
- Setting Flash Recovery Areas to the STANDBY_ARCHIVE_DEST Destination
- Sharing a Flash Recovery Area Between Primary and Standby Databases

---

Note: A primary database cannot transmit 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 8.4.4 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 Default Flash Recovery Area

If a flash recovery area has been configured and no local destinations are defined, Data Guard implicitly uses the `LOG_ARCHIVE_DEST_10` destination to refer to the default disk location for the flash recovery area and for storing the archived redo log files.

When the `LOG_ARCHIVE_DEST_10` destination is used for the flash recovery area, 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[1] 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_`*n* attributes, the `TEMPLATE` attribute of a `LOG_ARCHIVE_DEST_`*n* parameter will override all other specifications for the flash recovery area. If the `TEMPLATE` attribute is specified for a remote destination and

[1] Only the `QUOTA_SIZE` and `QUOTA_USED` attributes cannot be specified when defining a destination for the flash recovery area. This is because the amount of space allocated for the flash recovery area is defined with the `DB_RECOVERY_FILE_DEST_SIZE` parameter.

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 Setting Flash Recovery Areas to Other LOG_ARCHIVE_DEST_n Destinations

By default, if no local destinations are defined, flash recovery areas use the LOG_ARCHIVE_DEST_10 destination, but you can explicitly set up one or more other LOG_ARCHIVE_DEST_*n* destinations. 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 the flash recovery area to use another archival destination, 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 Setting Flash Recovery Areas to the STANDBY_ARCHIVE_DEST Destination

You can use a flash recovery area on a physical standby database by defining the `STANDBY_ARCHIVE_DEST` parameter. For example:

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

Note: Flash recovery area destinations specified with 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

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

5.3.1 Using Archiver Processes (ARCn) to Archive Redo Data

By default, log transport services use ARC*n* processes to archive the local online redo log files on the primary database *before* transmitting the redo data to remote standby destinations. Using ARC*n* processes for archival processing is described in the following topics:

- Initialization Parameters That Control ARCn Archival Behavior
- Default ARCn Archival Processing
- Nondefault ARCn Archival Processing

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.

5.3.1.1 Initialization Parameters That Control ARCn Archival Behavior

The `LOG_ARCHIVE_LOCAL_FIRST` initialization parameter, the `ARCH` attribute on the `LOG_ARCHIVE_DEST_`*n* parameter, and the `LOG_ARCHIVE_DEST_STATE_`*n* parameter control ARC*n* archival processing. The following sections describe setting these parameters to control archival processing.

Enabling Log Transport Services to Use ARCn Processes

The `ARCH` attribute of the `LOG_ARCHIVE_DEST_`*n* parameter enables log transport services to use ARC*n* processes to transmit redo data to archival destinations:

- If you specify the `ARCH` and `LOCATION` attributes on the `LOG_ARCHIVE_DEST_`*n* parameter, ARC*n* processes archive to a local destination.
- If you specify the `ARCH` and `SERVICE` attributes on the `LOG_ARCHIVE_DEST_`*n* parameter, ARC*n* processes transmit redo data to a remote destination.

Controlling When ARCn Processes Transmit Redo Data

The `LOG_ARCHIVE_LOCAL_FIRST` initialization parameter controls *when* the archiver processes (ARC*n*) transmit redo data to remote standby database destinations. The following table describes possible values for this parameter.

Value	Transmits Redo Data to the Remote Standby Destination. . .
`TRUE`	*After* the online redo log file is completely and successfully archived to at least one local destination. This is the default value. Section 5.3.1.2 provides more information about this default ARC*n* behavior.
`FALSE`	*At the same time* the online redo log file is archived to the local destinations. Section 5.3.1.3 provides more information about this ARC*n* behavior.

The following sections provide more information about the behavior of ARC*n* processing depending on the value of the `LOG_ARCHIVE_LOCAL_FIRST` initialization parameter.

5.3.1.2 Default ARCn Archival Processing

Figure 5–3 shows an example of the default archival processing in a Data Guard configuration. This configuration represents the default ARC*n* archival processing in a Data Guard configuration with a primary database located in Chicago and one physical standby database located in Boston. (This is the configuration that was created in Chapter 3.)

Archiving happens when a log switch occurs 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. (Section 5.6.2 describes how to configure standby redo log files.) Log apply services use Redo Apply (MRP process[1]) or SQL Apply (LSP process[2]) 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

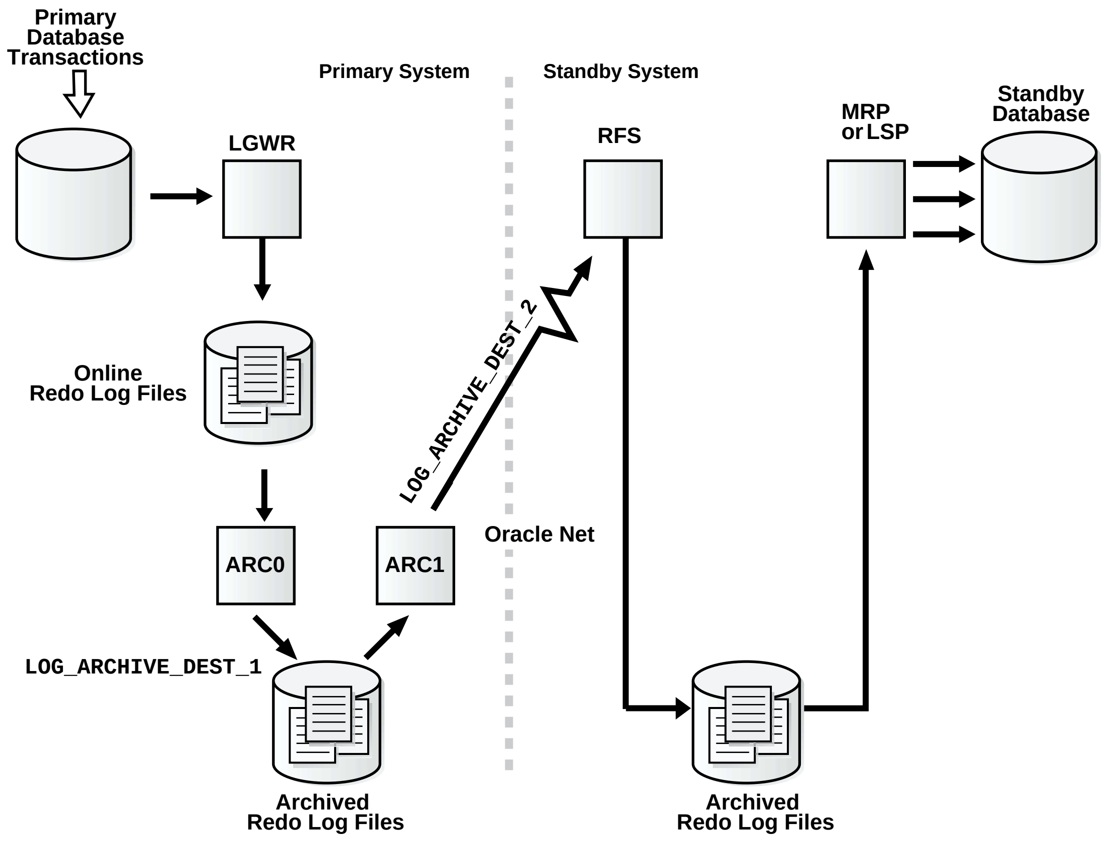
[1] The managed recovery process (MRP) applies archived redo log files to the physical standby database and can start additional parallel execution (P*nnn*) processes to balance workload.

[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.

database concurrently with the local destination. This behavior is useful when archiving to remote destinations that use a slow network connection, such as a long-distance wide area network (WAN). A benefit of the default ARC*n* archival behavior is that local archiving, and hence, processing on the primary database, is not affected by archiving to non-mandatory, remote destinations. It may be necessary to create more online redo log files, because it may take more time to recycle the online redo log files for reuse by the log writer process.

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 2).

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

5.3.1.3 Nondefault ARCn Archival Processing

To transmit redo data to the standby destination *at the same time* the online redo log file is being archived to the local online redo log files, set the `LOG_ARCHIVE_LOCAL_FIRST=FALSE` initialization parameter.

Note: Prior to release 10.1, the default ARC*n* archival behavior was to transmit redo data to the standby destination *at the same time* the online redo log file was being archived.

Example 5–5 shows the portion of the primary role initialization parameters with `LOG_ARCHIVE_LOCAL_FIRST=FALSE`. Note that specifying the `ARCH` attribute on the `LOG_ARCHIVE_DEST_`*n* parameter is optional, because this is the default archival setting.

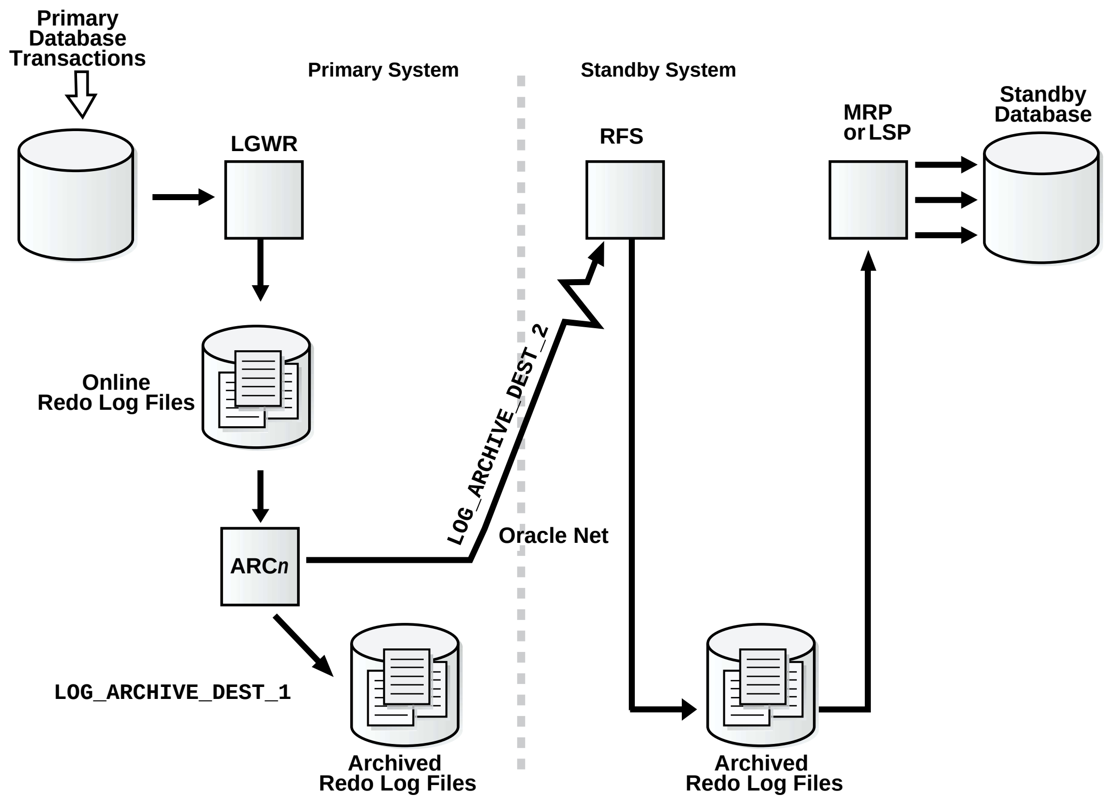
Example 5–5 Primary Database: Initialization Parameters for ARCn Archival

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

Figure 5–4 shows archival processing in a Data Guard configuration in which ARC*n* processes on the primary database transmit redo data to the remote destination at the same time the local online redo log file is being archived. In this configuration, archival operations occur on both the local and the remote standby destinations using redo data from the local online redo log files. This results in redo data being promptly dispatched to the remote standby database destination.

Specifying `LOG_ARCHIVE_LOCAL_FIRST=FALSE` is most useful for faster network connections, such as high-speed local area networks (LAN).

Figure 5–4 Archiving to Local and Remote Destinations at the Same Time



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

If you choose the LGWR process, it will transmit redo data to both the local and remote destinations as the redo is generated on the primary database. This section contains the following topics:

- LOG_ARCHIVE_DEST_n Attributes for LGWR Archival Processing
- LGWR SYNC Archival Processing
- LGWR ASYNC Archival Processing

Specifying the `LGWR` and `SYNC` attributes and configuring standby redo log files on at least one destination in a Data Guard configuration are required prerequisites for the maximum protection and maximum availability data protection modes. See Section 5.6 for information about the Data Guard data protection modes.

5.3.2.1 LOG_ARCHIVE_DEST_n Attributes for LGWR Archival Processing

You can optionally enable log transport services to use the LGWR process to concurrently transmit redo data to remote destinations at the same time the redo is written to the local online redo log files.

Using the LGWR process differs from the default ARC*n* 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 creates a new redo log file at the standby site that reflects the log sequence number (and size) of the current online redo log of the primary database. Then, as redo is generated at the primary database, it is also propagated to the remote destination. The propagation 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.

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

Enabling Log Transport Services to Use the LGWR Process

The `LGWR` attribute of the `LOG_ARCHIVE_DEST_`*n* parameter enables log transport services to use the LGWR process to transmit redo data to archival destinations You can specify the `LGWR` and `SERVICE` attributes on the `LOG_ARCHIVE_DEST_`*n* parameter to transmit redo data to a remote standby destination.

Specifying the Network Transmission Mode

By default, the LGWR process synchronously archives to the local online redo log files at the same time it transmits redo data to the remote destination. This is equivalent to specifying the `LGWR` and `SYNC` attributes on the `LOG_ARCHIVE_DEST_`*n* parameter:

- The `SYNC` attribute performs all network I/O synchronously, in conjunction with each write operation to the online redo log file. Transactions are not committed on the primary database until the redo data necessary to recover the transactions is received by the destination. Section 5.3.2.2 shows an example of synchronous network transmission in a Data Guard configuration.

  If you need to transmit redo data to multiple remote destinations, you can optionally specify `SYNC=PARALLEL` to initiate the network I/O to multiple destinations in parallel. When you specify both the LGWR and SYNC=PARALLEL attributes on the LOG_ARCHIVE_DEST_*n* parameter, 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.

If you do not specify either the SYNC or ASYNC attributes, the default is SYNC=PARALLEL.

- The ASYNC attribute performs all network I/O asynchronously and control is returned to the executing application or user immediately. When this attribute is specified, the LGWR process archives to the local online redo log file and submits the network I/O request to the network server (LNS*n*) process for that destination, and the LGWR process continues processing the next request without waiting for the network I/O to complete.

  If you specify the ASYNC attribute, you can specify a block count to determine the size of the SGA network buffer to be used. Block counts from 0 to 102,400 blocks are allowed. The ASYNC attribute allows the optional suffix value K to represent 1,000 (the value 1K indicates 1,000 512-byte blocks). In general, for slower network connections, use larger block counts. Section 5.3.2.3 shows an example of asynchronous network transmission in a Data Guard configuration.

  When the LGWR and ASYNC attributes are in effect, the LGWR process archives to the local online redo log file and submits the redo data to one or more LNS*n* processes that asynchronously transmit the redo data over the network. If log transport services transmit redo data to multiple remote destinations, the LNS*n* processes (one for each destination) initiate the network I/O to all of the destinations in parallel. See Chapter 12 for more information.

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 log transport services will revert to using the ARC*n* process to complete archival operations using the default (LOG_ARCHIVE_LOCAL_FIRST=TRUE) behavior. This behavior is described in Section 5.3.1.2.

5.3.2.2 LGWR SYNC Archival Processing

Example 5–6 shows the primary role LOG_ARCHIVE_DEST_*n* parameters that configure the LGWR process for synchronous network transmission. Note that specifying the SYNC attribute on the LOG_ARCHIVE_DEST_*n* parameter is optional, because synchronous network transmission is the default for LGWR archival processing. Also, the example specifies the NET_TIMEOUT=30 attribute to control 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 30 seconds, then the LGWR process returns an error message.

Example 5–6 Initialization Parameters for LGWR Synchronous Archival

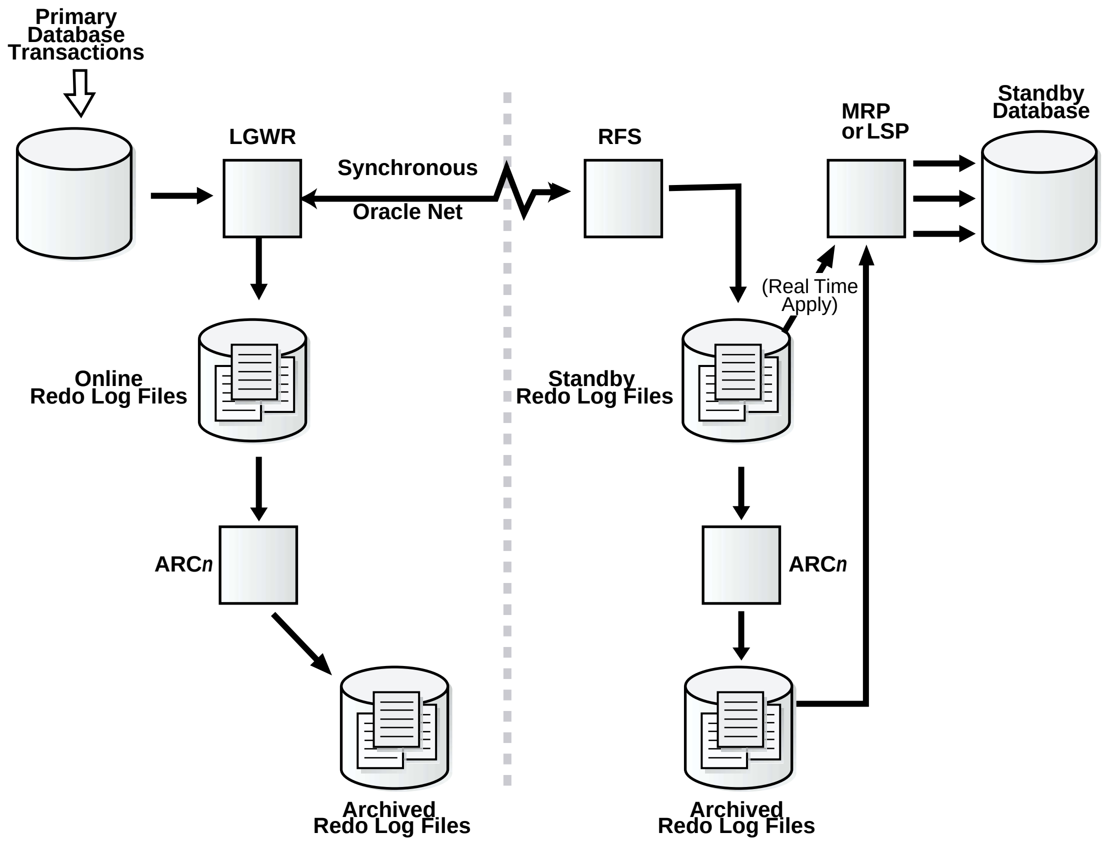
```
LOG_ARCHIVE_DEST_1='LOCATION=/arch1/chicago/
LOG_ARCHIVE_DEST_2='SERVICE=boston LGWR SYNC NET_TIMEOUT=30'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_STATE_2=ENABLE
```

Figure 5–5 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 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, log apply services use Redo Apply (MRP process) or SQL Apply (LSP process) to apply 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 5–5 LGWR SYNC Archival to a Remote Destination with Standby Redo Log Files



5.3.2.3 LGWR ASYNC Archival Processing

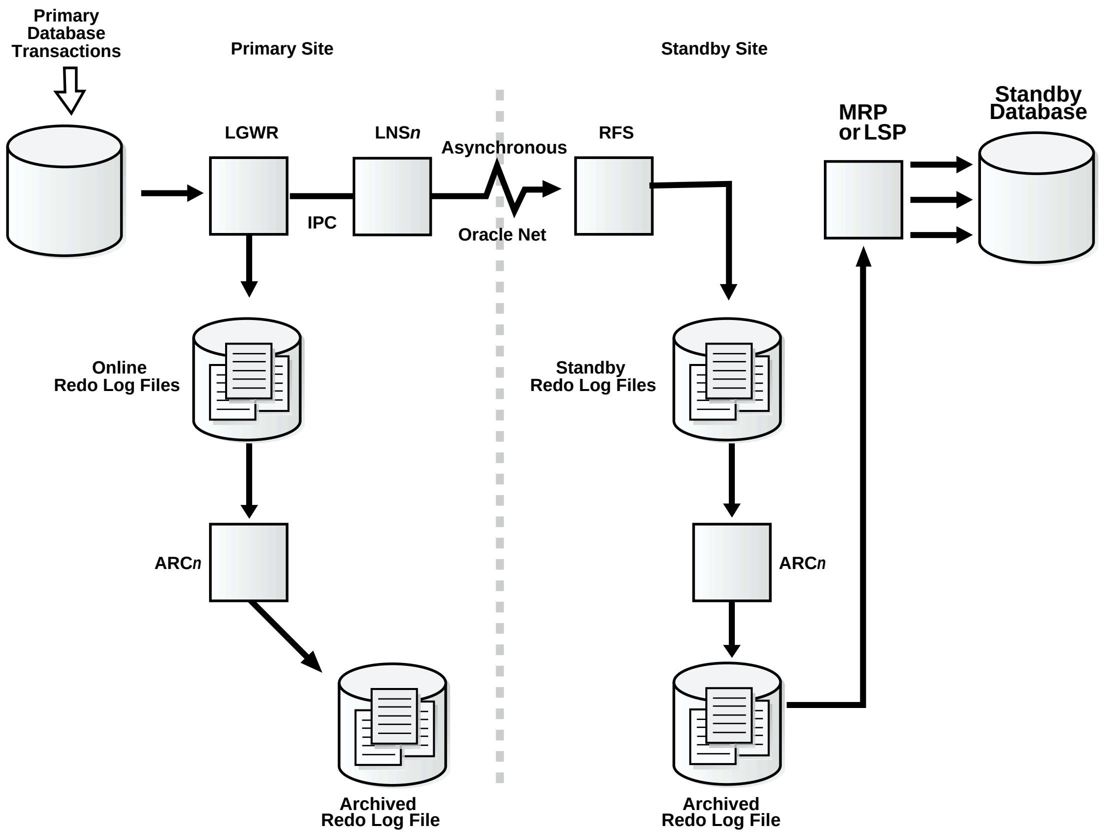
Example 5–7 shows the primary role LOG_ARCHIVE_DEST_*n* parameters that configure the LGWR process for asynchronous network transmission.

Example 5–7 Initialization Parameters for LGWR Asynchronous Archiving

```
LOG_ARCHIVE_DEST_1='LOCATION=/arch1/chicago/
LOG_ARCHIVE_DEST_2='SERVICE=boston LGWR ASYNC=61440'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_STATE_2=ENABLE
```

Figure 5–6 shows the LNS*n* process transmitting redo data over Oracle Net to the RFS process on the standby database. The LNS*n* and LGWR processes on the primary database use interprocess communication (IPC) to communicate.

Figure 5–6 LGWR ASYNC Archival with Network Server (LNSn) Processes



5.3.3 Providing for Secure Redo Data Transmission

Providing a secure environment should be a core requirement for any site supporting mission-critical applications, because a lack of security can directly affect availability. Data Guard provides a secure environment and prevents the possible tampering of redo data as it is being transferred to the standby database.

Log 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 Advanced Security Administrator's Guide.*

To provide for secure redo transmission, you need to set up every database in the Data Guard configuration to use a password file, and set the password for the SYS user identically on every system. To set up a secure environment 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.

Once you have performed these steps to set up security on every database in the Data Guard configuration, Data Guard transmits redo data only after the appropriate authentication checks using SYS credentials are successful.

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 log 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 10.1 for examples of the VALID_FOR attribute settings for various Data Guard configurations, and Chapter 12 for reference information about the VALID_FOR attribute.

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 10 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.

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
  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; log 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 will be deprecated in a future release. 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 and MAX_FAILURES 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.

Use the REOPEN attribute to determine if and when the ARC*n* 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 NOREOPEN, which prevents the destination from being retried after a failure occurs.

Use the MAX_FAILURE attribute to specify the maximum number of consecutive times that log transport services attempt to transmit redo data to a failed destination. You can use the REOPEN attribute, in conjunction with the MAX_FAILURE attribute, to limit the number of consecutive attempts that will be made to reestablish communication with a failed destination. Once the specified number of consecutive attempts is exceeded, the destination is treated as if the NOREOPEN attribute was specified.

The REOPEN attribute is required when you use the MAX_FAILURE attribute. Example 5–8 shows how to set a retry time of 60 seconds and limit retries to 3 attempts.

Example 5–8 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
- Configuring Standby Redo Log Files
- 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 5.6.3 for information about setting up the data protection mode.

5.6.1.1 Maximum Protection Mode

This protection mode guarantees 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 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 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 guarantees 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 `ASYNC` 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 `ASYNC` attributes.

5.6.2 Configuring Standby Redo Log Files

Standby redo log files are required for the maximum protection and maximum availability modes and highly recommended on all standby databases, because Data Guard can recover and apply more redo data from standby redo log files than from the archived redo log files alone.

You should plan the standby redo log configuration and create all required groups and members of groups either before or soon after 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.

Use the following steps to configure multiplexed standby redo log files:

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

The size of the current standby redo log file must exactly match (or be larger than) the size of the current primary database online redo log file. 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:

(maximum number of logfiles for each thread + 1) * 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 already set for the `MAXLOGFILES` and `MAXLOGMEMBERS` clauses on the SQL `CREATE DATABASE` statement will not limit the number of standby redo log file groups and number of members in each group 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 log files the next time there is a log switch on the primary database. Examples 5–9 and 5–10 show how to create a new group of standby redo log files, use the ALTER DATABASE statement with the ADD STANDBY LOGFILE GROUP clause.

Example 5–9 Adding a Standby Redo Log File Group to a Specific Thread

The following statement adds a new group of standby redo log files to a standby database and assigns them 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 5–10 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 standby redo log files are only used when the database is running in the standby role, Oracle recommends that you create standby redo log files on the primary database so that the primary database can switch over quickly to a standby role without the need for additional DBA intervention. Consider using the Oracle Enterprise Manager GUI to automatically configure standby redo log files 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. 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
```

5.6.3 Setting the Data Protection Mode of a Data Guard Configuration

To set up log 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 ASYNC when using LGWR process. SYNC if using ARCH process
Disk write option	AFFIRM	AFFIRM	AFFIRM or NOAFFIRM
Standby redo log required?	Yes	Yes	Optional, but 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 2 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 mode 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 3 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 4 Open the primary database.

If you performed Step 2 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 5 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 6 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 13 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
```

Log 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 or standby 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, log transport services store the fully qualified filenames in the standby database control file, and log apply services use 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 12 for information about the TEMPLATE and NOTEMPLATE 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 5–11 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 5–11 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
- Reassigning Standby Redo Log Groups to Threads

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 number of online redo log file groups, 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.

5.7.3.3 Reassigning Standby Redo Log Groups to Threads

If you used the `THREAD` clause to pre-assign a standby redo log group to a specific thread and later need to reassign the thread, first drop the standby redo log group (using the `DROP LOGFILE` clause) and add it again using the `ALTER DATABASE ADD STANDBY LOGFILE THREAD` *n* statement.

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 log 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.

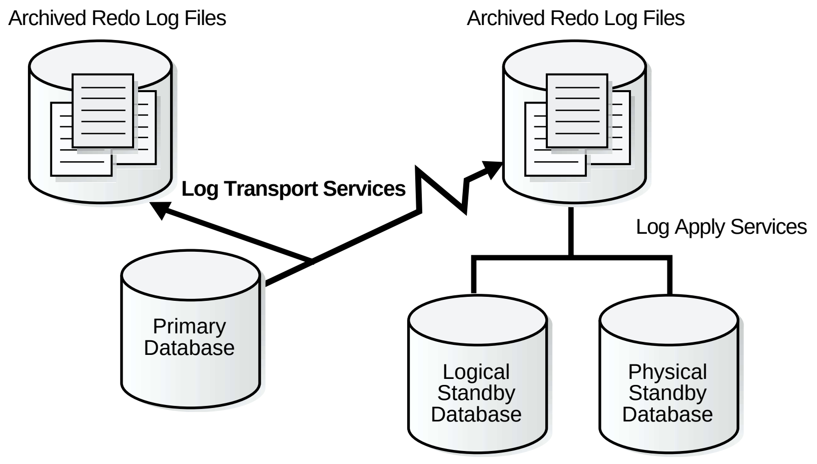
See *Oracle Database Reference* for more details about the CONTROL_FILE_RECORD_KEEP_TIME initialization parameter and *Oracle Database Backup and Recovery Advanced User's Guide.*

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.

In these situations, although the ARC*n* processes do not physically archive the redo data to each standby destination, the standby destinations need to know the location of the archived redo log files. This allows the standby database to access the archived redo log files when they become available for application by log apply services. You must specify an archiving destination as being dependent on the success or failure of another (parent) destination.

5.8 Managing Archive Gaps

An **archive gap** can occur on the standby system when it is 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 10.8 for a scenario that shows how to resolve a gap manually.

Note: Prior to Oracle Database 10*g* 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) Process to Resolve Archive Gaps

The fetch archive log (FAL) process resolves 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

FAL_SERVER	This parameter 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**
		FAL_SERVER=*net_service_name*
		Example
		FAL_SERVER=standby2_db,standby3_db

FAL_CLIENT	This parameter specifies the network service name that the FAL server should use to connect to the standby database.	**Syntax**
		FAL_CLIENT=*net_service_name*
		Example
		FAL_CLIENT=standby1_db

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 Log 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 current archived redo log file sequence numbers.

Enter the following query on the primary database to determine the current archived redo log file sequence numbers:

```
SQL> SELECT THREAD#, SEQUENCE#, ARCHIVED, STATUS FROM V$LOG
  2> WHERE STATUS='CURRENT';
```

Step 2 Determine the most recent archived redo log file.

Enter the following query at the primary database to determine which archived redo log file contains the most recently transmitted redo data:

```
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 at 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 E for complete details and examples.

5.9.2 Monitoring the Performance of Log Transport Services

This section describes the wait events that monitor the performance of the log 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=NOPARALLEL Wait Events
- LGWR ASYNC Wait Events
- Network Server (LNSn) Wait Events

5.9.2.1 ARCn Process Wait Events

For ARC*n* archival processing, Table 5–3 shows 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 5.3.1 for information about ARC*n* 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 ARC*n* processes to spawn an RFS connection.

Table 5–3 (Cont.) Wait Events for Destinations Configured with the ARCH Attribute

Wait Event	Monitors the Amount of Time Spent By . . .
ARCH wait on SENDREQ	All ARC*n* 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 ARC*n* processes to delete an RFS connection.

5.9.2.2 LGWR SYNC=NOPARALLEL Wait Events

For `LGWR SYNC=NOPARALLEL` archival processing, Table 5–4 shows 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 acknowledgement 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 ATTACH	All LGWR processes to spawn an RFS connection.
LGWR wait on SENDREQ	All LGWR processes to write the received redo data to disk as well as open and close the remote archived redo log files.
LGWR wait on DETACH	All LGWR processes to delete an RFS connection.

5.9.2.3 LGWR ASYNC Wait Events

For `LGWR ASYNC` archival processing, Table 5–5 shows 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 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.

Table 5–5 (Cont.) Wait Events for Destinations Configured with the LGWR ASYNC

Wait Event	Monitors the Amount of Time Spent By . . .
LGWR wait on full LNS buffer	The LGWR process waiting for the network server (LNS) to free up ASYNC buffer space. If buffer space has not been freed in a reasonable amount of time, availability of the primary database is not compromised by allowing the ARC*n* process to transmit the redo data. **Note:** This wait event is not relevant for destinations configured with the `LGWR SYNC=NOPARALLEL` attributes.

5.9.2.4 Network Server (LNSn) Wait Events

When either the `LGWR` and `ASYNC` attributes or the `LGWR` and `SYNC=PARALLEL` attributes are in effect, the LGWR process archives to the local online redo log file and submits the redo data to one or more LNS*n* processes (one for each destination) that asynchronously transmit the redo data over the network. Table 5–6 shows the wait events that monitor the time it takes for the LGWR and LNS*n* processes to communicate over interprocess communication (IPC) channels. See Section 5.3.2.3 for more information about configurations using the LGWR and LNS*n* processes.

Table 5–6 Wait Events for LGWR ASYNC or LGWR SYNC=PARALLEL Attributes

Wait Event	Monitors the Amount of Time Spent By . . .
LGWR wait on LNS	The LGWR process waiting to receive messages on IPC channels from the network server.
LNS wait on LGWR	The network server waiting to receive messages on IPC channels from the LGWR process.
LGWR-LNS wait on channel	The LGWR process or the network server processes waiting to receive messages on IPC channels.

6 Log Apply Services

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

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 recovering 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 optionally to standby redo log files. However, if you use standby redo log files, you can optionally 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 transactionally current 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, "Using a Standby Database That Is Open for Read-Only 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, as described in Section 9.1.2). 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:

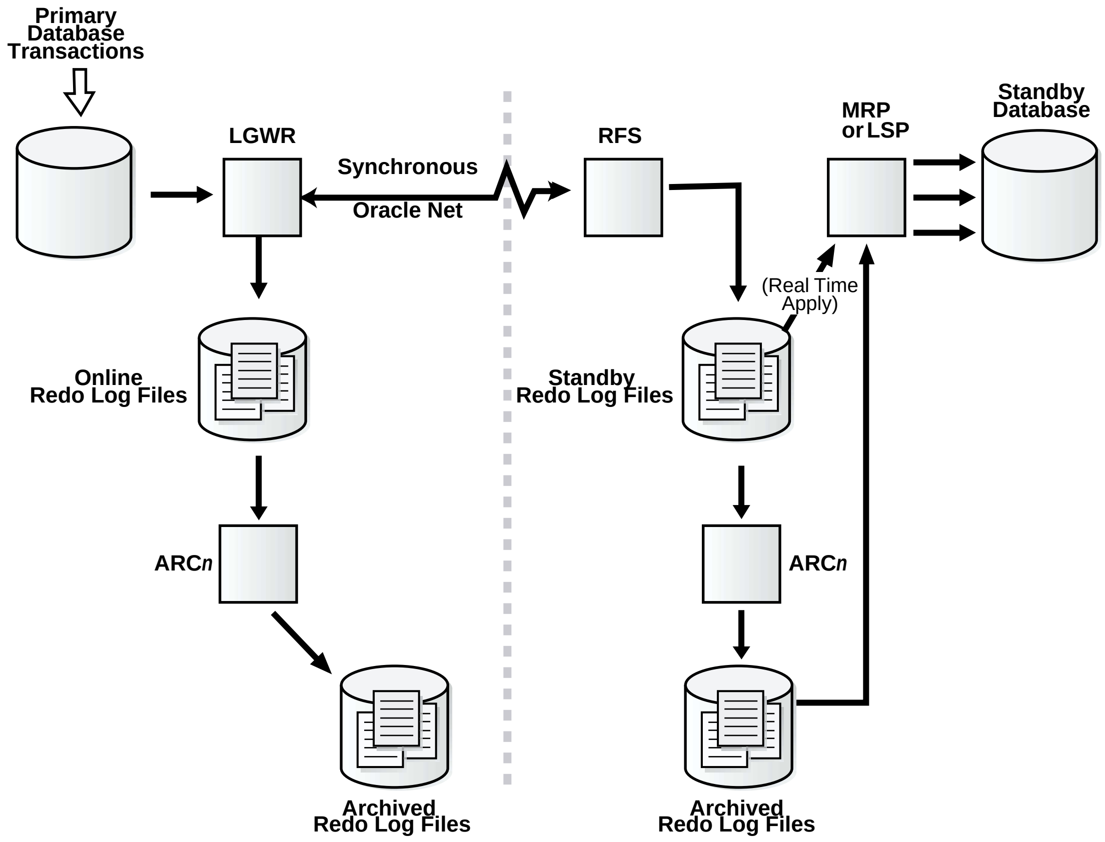
- Using Real-Time Apply to Apply Redo Data Immediately
- Specifying a Time Delay for the Application of Archived Redo Log Files

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. (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



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.

To determine if real-time apply is enabled, query the `RECOVERY_MODE` column in the `V$ARCHIVE_DEST_STATUS` view. It will display `MANAGED REAL-TIME APPLY` when real-time apply is enabled.

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, as follows:

- On the primary database and physical standby databases, use the `DELAY=`*minutes* attribute of the `LOG_ARCHIVE_DEST_`*n* initialization parameter to delay applying archived redo log files to the standby database. The default setting for this attribute is `NODELAY`. If you specify the `DELAY` attribute without specifying a value, then the default delay interval is 30 minutes.
- On logical standby databases, use the `DBMS_LOGSTDBY.APPLY_SET` procedure.

Setting up a time delay on a standby database supersedes any time delay specified on the primary database. For example:

```
SQL> RECOVER MANAGED STANDBY DATABASE DELAY <minutes>
```

In a configuration with multiple standby databases, setting a time lag on more than one standby database can be very useful. For example, you can set up a configuration where each standby database is maintained in varying degrees of synchronization with the primary database.

Canceling a Time Delay

You can cancel a specified delay interval as follows:

- On the primary database and physical standby databases, use the `NODELAY` keyword of the `RECOVER MANAGED STANDBY DATABASE` clause:

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

- On logical standby databases, specify the following PL/SQL command:

  ```
  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 10.5, "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
- *PL/SQL Packages and Types Reference* for logical standby databases using the `DBMS_LOGSTDBY.APPLY_SET` procedure

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

As an alternative to the apply delay configuration option, you can use Flashback Database to protect against 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 *Oracle Database Backup and Recovery Advanced User's Guide* for more information about enabling and using Flashback Database.

See Chapter 10 for scenarios showing how to use Data Guard with Flashback Database, and *Oracle Database Backup and Recovery Advanced User's Guide* 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. Also, 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
- Starting Real-Time Apply
- Stopping Log Apply Services
- Monitoring Log Apply Services 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.

- To start a foreground session that recovers a database using the archived redo log on the physical standby database, issue the SQL statement:

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

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

- To start a background process that recovers a database using the archived redo log on the physical standby database, you *must* use 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.

6.3.2 Starting Real-Time Apply

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;
```

6.3.3 Stopping Log Apply Services

To stop Redo Apply or real-time apply, issue the following SQL statement in another window:

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

6.3.4 Monitoring Log Apply Services on Physical Standby Databases

To monitor the status of the archived redo log and obtain information about 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$MANAGED_STANDBY Fixed View
- Accessing the V$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

See *Oracle Database Reference* for complete reference information about views.

6.3.4.1 Accessing the V$MANAGED_STANDBY Fixed View

Query the physical standby database to monitor log apply and log 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 the redo log file with sequence number 947. The output also shows Redo Apply when it 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.

6.3.4.2 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. This might indicate a single recovery process is unable to keep up with the volume of the archived redo log files being received. Using the `PARALLEL` option might be a solution.

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` 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
```

6.3.4.3 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, because 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.

6.3.4.4 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.

6.3.4.5 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/sorl2.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/sorl1.log'
Attempt to start background Managed Standby Recovery process
Media Recovery Log /oracle/arch/arch_1_9.arc

10 rows selected.
```

6.4 Applying Redo Data to Logical Standby Databases

Log apply services convert the data from the archived redo log or standby redo log into 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
- Starting Real-time Apply
- Stopping Log Apply Services on a Logical Standby Database
- Monitoring Log Apply Services for Logical Standby Databases

6.4.1 Starting SQL Apply

To start SQL Apply, start the logical standby database and issue the following statement to recover redo data from archived redo log files on the logical standby database:

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

6.4.2 Starting Real-time Apply

To start real-time apply on the logical standby database to immediately recover 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;
```

6.4.3 Stopping Log Apply Services 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;
```

6.4.4 Monitoring Log Apply Services for Logical Standby Databases

To monitor the status of archived redo log files and obtain information about SQL Apply, query the fixed views described in this section. You can also monitor the standby database using the Oracle Enterprise Manager GUI. See Appendix A, "Troubleshooting Data Guard" and *Oracle Data Guard Broker*.

This section contains the following topics:

- Accessing the DBA_LOGSTDBY_EVENTS View
- Accessing the DBA_LOGSTDBY_LOG View
- Accessing the DBA_LOGSTDBY_PROGRESS View

- Accessing the V$LOGSTDBY Fixed View
- Accessing the V$LOGSTDBY_STATS Fixed View

Also, see the discussion of the V$ARCHIVE_DEST_STATUS fixed view in Section 6.3.4.2 and *Oracle Database Reference* for complete reference information about views.

6.4.4.1 Accessing the DBA_LOGSTDBY_EVENTS View

If SQL Apply should stop unexpectedly, the reason for the problem is shown in this view.

Note: Errors that cause SQL Apply to stop are recorded in the events table (unless there is insufficient space in the system tablespace). 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, 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 statements 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_TIME, COMMIT_SCN;

EVENT_TIME         STATUS
------------------------------------------------------------------------------
EVENT

-------------------------------------------------------------------------------
23-JUL-02 18:20:12 ORA-16111: log mining and apply setting up
23-JUL-02 18:20:12 ORA-16128: User initiated shut down successfully completed
23-JUL-02 18:20:12 ORA-16112: log mining and apply stopping
23-JUL-02 18:20:23 ORA-16111: log mining and apply setting up
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 mytable (one number, two varchar(30))
```

```
23-JUL-02 20:22:55 ORA-16205: DDL skipped due to skip setting create database
link mydblink

8 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.

6.4.4.2 Accessing the DBA_LOGSTDBY_LOG View

The DBA_LOGSTDBY_LOG view provides dynamic information about what is happening to SQL Apply. This view is helpful when you are diagnosing performance problems when SQL Apply is applying archived redo log files to the logical standby database, and it can be helpful for other problems.

For example:

```
SQL> COLUMN DICT_BEGIN FORMAT A10;
SQL> SELECT FILE_NAME, SEQUENCE#, FIRST_CHANGE#, NEXT_CHANGE#,
  2> TIMESTAMP, DICT_BEGIN, DICT_END, THREAD# AS THR# FROM DBA_LOGSTDBY_LOG
  3> 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
```

The output from this query shows 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.

6.4.4.3 Accessing the DBA_LOGSTDBY_PROGRESS View

This view shows the state of the LSP process and information about the SQL transactions that were executed on the logical standby database. To quickly determine if all redo from the log file was applied, issue the following query on the logical standby database:

```
SQL> SELECT APPLIED_SCN, NEWEST_SCN FROM D BA_LOGSTDBY_PROGRESS;

APPLIED_SCN NEWEST_SCN
----------- ----------
     211301     211357
```

If the APPLIED_SCN matches the NEWEST_SCN, then all available log information was applied. To determine how much progress was made through the available log files, query the DBA_LOGSTDBY_LOG view, as shown in the following example:

```
SQL> ALTER SESSION SET NLS_DATE_FORMAT  = 'DD-MON-YY HH24:MI:SS';
Session altered.

SQL> SELECT SEQUENCE#, FIRST_TIME, APPLIED
  2   FROM DBA_LOGSTDBY_LOG
  3  ORDER BY SEQUENCE#;

 SEQUENCE# FIRST_TIME         APPLIED
---------- ------------------ -------
        24 23-JUL-02 18:19:05 YES
        25 23-JUL-02 18:19:48 YES
        26 23-JUL-02 18:19:51 YES
        27 23-JUL-02 18:19:54 YES
        28 23-JUL-02 18:19:59 YES
        29 23-JUL-02 18:20:03 YES
        30 23-JUL-02 18:20:13 YES
        31 23-JUL-02 18:20:18 YES
        32 23-JUL-02 18:20:21 YES
        33 23-JUL-02 18:32:11 YES
        34 23-JUL-02 18:32:19 CURRENT
        35 23-JUL-02 19:13:20 CURRENT
        36 23-JUL-02 19:13:43 CURRENT
        37 23-JUL-02 19:13:46 CURRENT
        38 23-JUL-02 19:13:50 CURRENT
        39 23-JUL-02 19:13:54 CURRENT
        40 23-JUL-02 19:14:01 CURRENT
        41 23-JUL-02 19:15:11 NO
        42 23-JUL-02 19:15:54 NO
19 rows selected.
```

In the previous query, the computed APPLIED column displays YES, CURRENT, NO. The log files with YES were completely applied and those files are no longer needed by the logical standby database. The log files with CURRENT contain information that is currently being worked on. Because logical standby applies transactions, and because transactions span log files, it is common for SQL Apply to be applying changes from multiple log files. For logs with NO, information from those files is not being applied. Although it is possible that the files might have been open and read.

6.4.4.4 Accessing the V$LOGSTDBY Fixed View

To inspect the process activity for SQL Apply, query the V$LOGSTDBY fixed view on the logical standby database. This view provides information about the processes that are reading redo data and applying it to logical standby databases. For example:

```
SQL> COLUMN STATUS FORMAT A50
SQL> COLUMN TYPE FORMAT A12
SQL> SELECT TYPE, HIGH_SCN, STATUS FROM V$LOGSTDBY;

TYPE           HIGH_SCN STATUS
------------ ---------- --------------------------------------------------
COORDINATOR             ORA-16117: processing
READER                  ORA-16127: stalled waiting for additional transact
                        ions to be applied

BUILDER          191896 ORA-16116: no work available
PREPARER         191902 ORA-16117: processing
ANALYZER         191820 ORA-16120: dependencies being computed for transac
                        tion at SCN 0x0000.0002ed4e

APPLIER          191209 ORA-16124: transaction 1 16 1598 is waiting on ano
                        ther transaction

APPLIER          191205 ORA-16116: no work available
APPLIER          191206 ORA-16124: transaction 1 5 1603 is waiting on anot
                        her transaction

APPLIER          191213 ORA-16117: processing
APPLIER          191212 ORA-16124: transaction 1 20 1601 is waiting on ano
                        ther transaction

APPLIER          191216 ORA-16124: transaction 1 4 1602 is waiting on anot
                        her transaction
11 rows selected
```

The previous query displays one row for each process involved in reading and applying archived redo log files. The different processes perform different functions as described by the TYPE column. The HIGH_SCN column is a progress indicator. As long as it keeps changing, from query to query, you know progress is being made. The STATUS column gives a text description of activity.

6.4.4.5 Accessing the V$LOGSTDBY_STATS Fixed View

The V$LOGSTDBY_STATS fixed view provides a collection of state and statistical information for SQL Apply. Most options have default values, and this view displays what values are currently in use. It also provides statistical information that helps indicate progress. Issue the following query to view database state information:

```
SQL> COLUMN NAME FORMAT A35
SQL> COLUMN VALUE FORMAT A35
SQL> SELECT NAME, VALUE FROM V$LOGSTDBY_STATS
  2> WHERE NAME LIKE 'coordinator%' or NAME LIKE 'transactions%';

NAME                                VALUE
----------------------------------- -----------------------------------
coordinator state                   APPLYING
transactions ready                  7821
transactions applied                7802
coordinator uptime                  73
```

This query shows how long SQL Apply was running and how many transactions were applied in that time. It also shows how many transactions are available to be applied, indicating that more work is necessary.

6.5 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 Network 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_ASYNCH_IO to TRUE (the default). The DISK_ASYNCH_IO parameter controls whether or not network I/O to datafiles is asynchronous. Asynchronous

I/O should significantly reduce database file parallel reads and should improve overall recovery time.

7 Role Management

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, or when performing hardware or software maintenance, you must transition the role of the 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, in advance, the role-management options available to you in response to a planned or unplanned primary database outage.

This chapter describes Data Guard role management services and operations that allow you to change and manage roles of the databases 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

See *Oracle Data Guard Broker* for information about using Oracle Data Guard broker's distributed management framework to automate the switchover and failover processes into a single command. The Data Guard broker provides GUI and command-line interfaces that automate and centralize the creation, maintenance, and monitoring of Data Guard configurations.

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 two role transition operations:

- **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**

  Transitions 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. After a failover, the failed database no longer participates in the Data Guard configuration.

Section 7.1.1 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 7.1.2 and Section 7.1.3, respectively.

Note: Oracle Data Guard switchovers and failovers are not invoked automatically. You must initiate a switchover or failover manually using a SQL statement or a Data Guard broker interface.

7.1.1 Which Role Transition to Use

During any role transition, the amount of downtime required to complete the operation, the potential for data loss, and the effects on other standby databases in the configuration are determined by:

- The state of the primary database just before the transition
- The state of the standby database selected for the role transition at the time of the transition
- If the selected standby database is configured as a physical standby database or a logical standby database
- If the role transition is a switchover or a failover
- How much redo data remains to be applied to the selected standby database

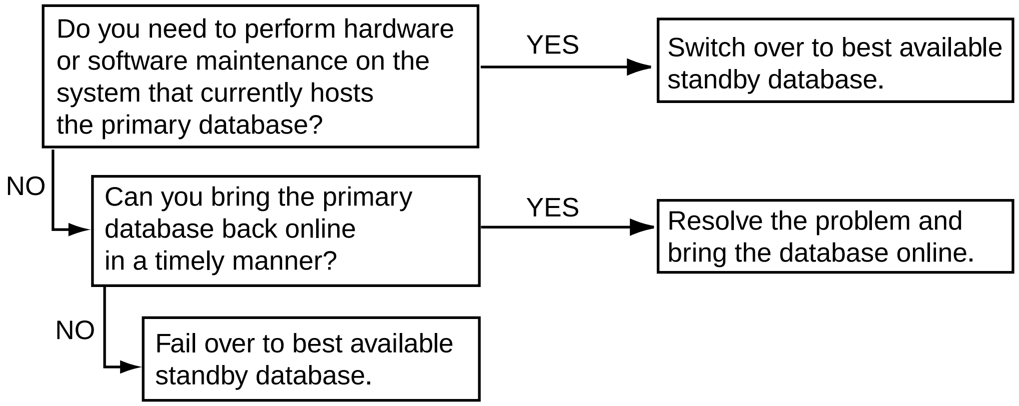
- Whether or not a standby redo log is configured on the standby database
- Whether or not redo log files were previously created for the target standby database

The goal is to perform the role transition as quickly as possible with little or no data loss.

Note: The time required to complete a role transition is minimized when the real-time apply feature is enabled and active, as described in Section 6.2.1. Real-time apply allows log apply services to recover and apply redo data from standby redo log files at the same time these files are receiving new redo data from the primary database. This ensures the lag between the standby database and the primary database is as small as possible.

The decision tree presented in Figure 7–1 can help you choose the role transition that best minimizes downtime and risk of data loss.

Figure 7–1 Role Transition Decision Tree


Do you need to perform hardware or software maintenance on the system that currently hosts the primary database?
YES
Switch over to best available standby database.
NO
Can you bring the primary database back online in a timely manner?
YES
Resolve the problem and bring the database online.
NO
Fail over to best available standby database.

In general, consider if it would be faster to repair the primary database than to perform a role transition. If you can repair the primary database, you also do not have to reconfigure client applications to connect to a new database. However, if the repair operation results in any data loss, you might be able to flash back the standby databases as described in Section 10.4. If you do not have Flashback Database enabled, you might need to re-create all other standby databases in the configuration from a backup copy of the repaired primary database.

If you decide that a role transition is appropriate and the configuration contains one or more physical standby databases, Oracle recommends that you perform the role transition using the best available physical standby database. Role transitions involving a logical standby database:

- Can result in data loss if the logical standby database is configured to maintain only a subset of the data present in the primary database
- Require that any existing physical standby databases be re-created from a copy of the new primary database to continue to participate in the Data Guard configuration after the role transition

See Section 10.2 for information about how to choose the best available physical or logical standby database

Once you determine the type of role transition you want to perform, proceed to one of the following sections:

- For switchovers, see Section 7.1.2.
- For failovers, see Section 7.1.3.

7.1.2 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 Section 9.2).

A switchover takes place in two phases. In the first phase, the existing primary database is transitioned to a standby role. In the second phase, a standby database is transitioned to the primary role.

Figure 7–2 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 7–2 Data Guard Configuration Before Switchover

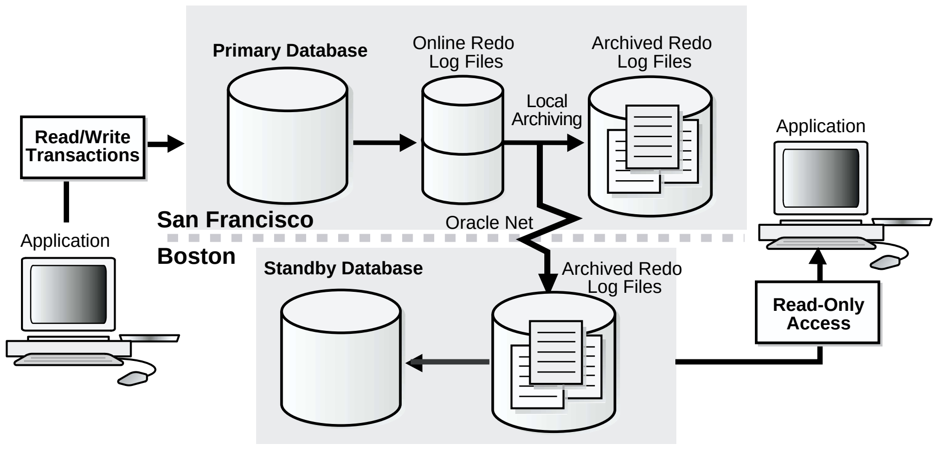


Figure 7–3 shows the Data Guard environment after the original primary database was switched over to a standby database, but before the original standby database has become the new primary database. At this stage, the Data Guard configuration temporarily has two standby databases.

Figure 7–3 Standby Databases Before Switchover to the New Primary Database

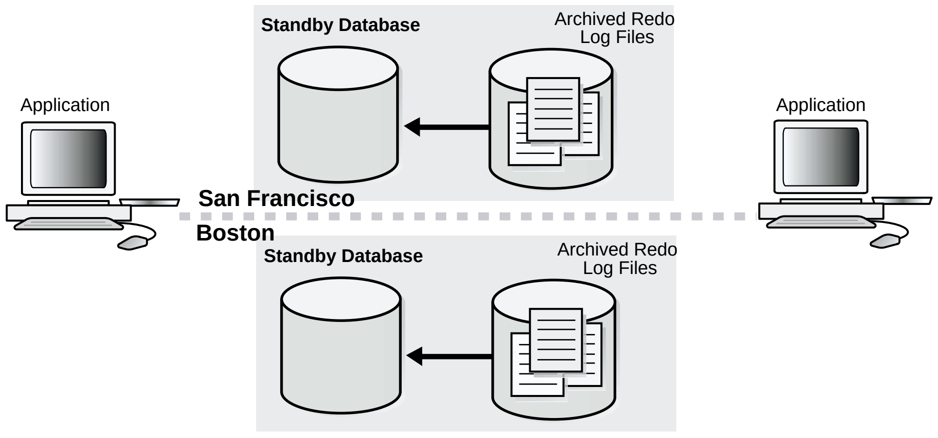
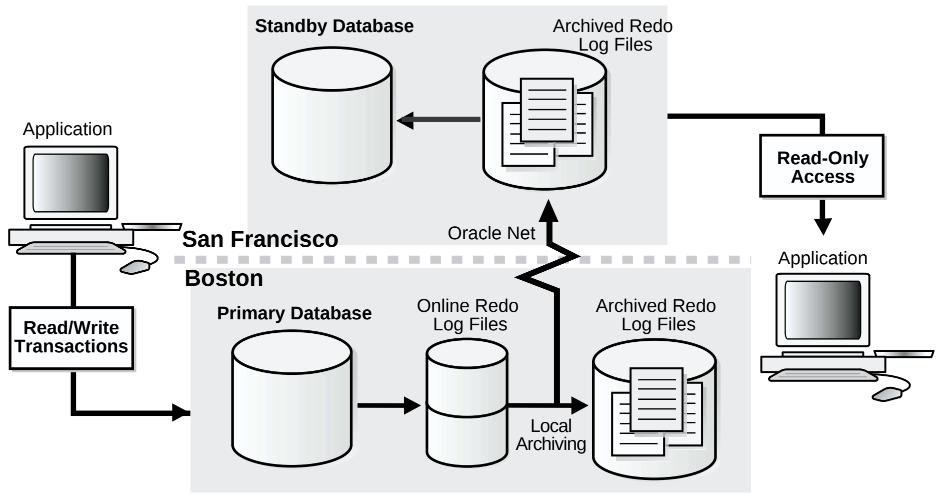


Figure 7–4 shows the Data Guard environment after a switchover took place. The original standby database became the new primary database. The primary database is now in Boston, and the standby database is now in San Francisco.

Figure 7–4 Data Guard Environment After Switchover



7.1.2.1 Preparing for a Switchover

Although switchovers can be performed between the primary database and either a logical or a physical standby database in the Data Guard configuration, a physical standby database is preferred (as described in Section 7.1.1). To minimize downtime, carefully plan each switchover so that the primary and standby databases involved have as small a transactional lag as possible. Also, consider using the Data Guard broker to automate and simplify the switchover procedure into one easy step. See *Oracle Data Guard Broker* for more information.

Before starting a switchover:

- Verify the initialization parameters for each database are configured correctly. See Chapter 3 and Chapter 4 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.

Note: If you do not use the Data Guard broker, you must define the LOG_ARCHIVE_DEST_*n* and LOG_ARCHIVE_DEST_STATE_*n* parameters on all standby sites so that when a switchover or failover occurs, all of the standby sites continue to receive redo data from the *new* primary database. See Section 5.4.1 and Chapter 12 for information about using the LOG_ARCHIVE_DEST_*n* VALID_FOR attribute to define role-based destinations in preparation for future role transitions.

Configurations that you set up with the Data Guard broker (command-line interface or GUI) handle the LOG_ARCHIVE_DEST_*n* and LOG_ARCHIVE_DEST_STATE_*n* definitions automatically, including defining the LOG_ARCHIVE_DEST_*n* parameters to point back to the primary database and all of the other standby databases.

- Verify there is network connectivity between the primary and standby locations.

  Each location in the Data Guard configuration should have connectivity through Oracle Net to the primary database and to all other associated standby databases.

- 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. See Section 3.2.6 (physical standby databases) and Section 4.2.4 (logical standby databases) for information about creating the temporary file on the standby database.
- Remove any redo data application delay in effect on the standby database that will become the new primary database.
- Verify there are no active users connected to the databases.
- Verify all but one primary instance and one standby instance in a Real Application Clusters configuration are shut down.

  For a Real Application Clusters database, only one primary instance and one standby instance can be online during the switchover. Shut down all other instances before starting the switchover. Then, after the switchover completes, bring these instances back online.

Note: Even though only one standby instance is open during the switchover, all of the standby database instances will be automatically transitioned to their new role correctly.

- 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 transition to the primary role must be mounted before you begin the switchover. Ideally, the physical standby database will also be actively recovering archived redo log files 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 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 for more information about SQL Apply.

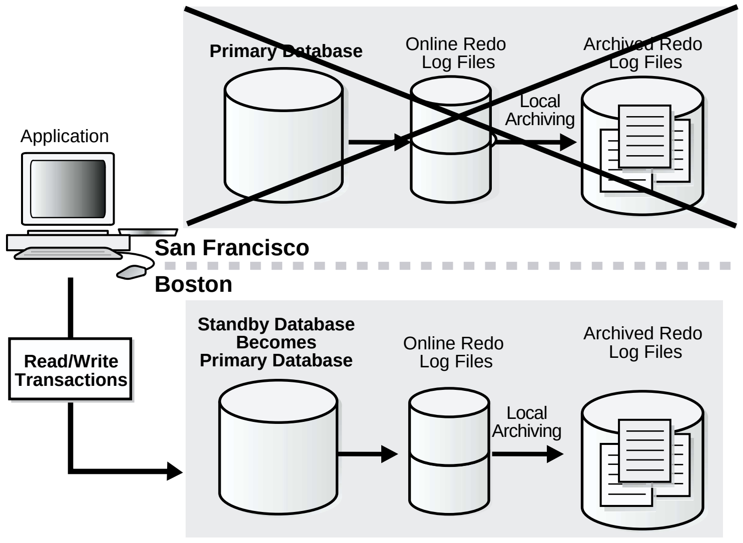
For switchovers involving a physical standby database, see Section 7.2.1. For switchovers involving a logical standby database, see Section 7.3.1. If you configured your environment using Oracle Data Guard broker distributed management framework, refer instead to *Oracle Data Guard Broker* for information about how to use the Switchover wizard to automate the switchover process.

7.1.3 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 7–5 shows the result of a failover from a primary database in San Francisco to a physical standby database in Boston.

Figure 7–5 Failover to a Standby Database



7.1.3.1 Preparing for 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 by following the steps described in this section. Consider using the Data Guard broker to automate and simplify the failover procedure into one easy step. See *Oracle Data Guard Broker* for more information.

Before initiating a failover, perform the following steps:

- Verify the initialization parameters for the surviving databases in the Data Guard configuration are configured correctly. See Chapter 3 for information about how to configure initialization parameters on the primary and physical standby databases so that your Data Guard configuration operates properly after the role transition.

Note: If you do not use the Data Guard broker, you must define the `LOG_ARCHIVE_DEST_`*n* and `LOG_ARCHIVE_DEST_STATE_`*n* parameters on all standby sites so that when a switchover or failover occurs, all of the standby sites continue to receive redo data from the new primary database. See Section 5.4.1 and Chapter 12 for information about using the `LOG_ARCHIVE_DEST_`*n* `VALID_FOR` attribute to define role-based destinations in preparation for future role transitions.

Configurations that you set up with the Data Guard broker (command-line interface or GUI) handle the `LOG_ARCHIVE_DEST_`*n* and `LOG_ARCHIVE_DEST_STATE_`*n* definitions automatically, including defining the `LOG_ARCHIVE_DEST_`*n* parameters to point back to the primary database and all the other standby databases.

- Verify each remaining location in the Data Guard configuration has network connectivity through Oracle Net to the database that will become the new primary database and to all other associated standby databases.
- 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. See Section 3.2.6 (physical standby databases) and Section 4.2.4 (logical standby databases) for information about creating the temporary file on the standby database.
- Remove any redo data application delay that is in effect on the standby database that will become the new primary database.
- If the standby database that will become the new primary database is a Real Application Clusters database, shut down all but one standby instance before starting the failover. Then, after the failover completes, bring the other instances back online. For a Real Application Clusters database, only one standby instance can be active during the 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 irreversibly 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 for information about how to verify a physical standby database is operating correctly.
- See Section 4.2.5 for information about how to verify a logical standby database is operating correctly.

To perform a failover involving a physical standby database, see Section 7.2.2. To perform a failover involving a logical standby database, see Section 7.3.2.

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 that changes roles between a primary database and a physical standby database. A switchover must be initiated on the current primary database and completed on the target standby database. The following steps describe how to perform the switchover.

On the current primary database:

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 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 transition 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 7–3).

On the target physical standby database:

Step 4 Verify the switchover status in the V$DATABASE view.

After you transition 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 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 mounted in one of these modes so that the primary database switchover request can be coordinated. After you mount the standby database in an appropriate mode, issue the following SQL statement on the physical standby database that you want to transition to the primary role:

```
SQL> ALTER DATABASE COMMIT TO SWITCHOVER TO PRIMARY;
```

Also see Chapter 3 for information about manually adding redo log files when creating a physical standby database.

Step 6 Shut down and restart the target standby database.

Shut down the target standby database and restart it using the appropriate initialization parameters for the primary role:

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

The target physical standby database is now transitioned to the primary database role.

> **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.

On the new physical standby database and on all other standby databases:

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 for more information about how to configure and start log apply services.

On the new primary database:

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.

Before starting the failover, perform as many of the steps documented in Section 7.1.3.1 as possible to prepare the selected standby database for the failover operation, then proceed to Section 7.2.2.1 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 causes data loss.

7.2.2.1 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, 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. This 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 the failover operation on the target physical standby database.

If the target physical standby database has standby redo log files configured, issue the following statement to initiate the failover:

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

If the target physical standby database does *not* have standby redo log files configured, include the `FINISH SKIP STANDBY LOGFILE` clause:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE
  2> FINISH SKIP STANDBY LOGFILE;
```

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).

Step 5 Convert the physical standby database to the primary role.

Once the SQL `ALTER DATABASE RECOVER MANAGED STANDBY DATABASE...FINISH` statement completes successfully, transition 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 is transitioned 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.

On the new primary database:

Step 6 Shut down and restart the new primary database.

To complete the failover, you need to shut down the new primary database and restart it in read / write mode using the proper traditional initialization parameter file (or server parameter file) for the primary role:

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

See Chapter 3 and Chapter 4 for information about how to configure initialization parameters on both the primary and standby databases so that your Data Guard configuration operates properly after a role transition.

Step 7 Optionally, back up the new primary database.

Optionally, before issuing the STARTUP statement, you might want to perform a closed back up of the new primary database. In place of a closed backup, instead consider performing an open backup of the database after issuing the STARTUP statement. Although performing a backup immediately is not required, it is a recommended 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 optionally 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 10.3, "Using Flashback Database After a Failover".

Note: You must have already enabled Flashback Database on the old primary database before the failover. See *Oracle Database Backup and Recovery Advanced User's Guide* 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, "Creating a Physical Standby Database"or Section 4.2, "Creating a Logical Standby Database".

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 7.2.1, "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. The following steps describe how to perform the switchover.

On the primary database:

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
```

A value of TO STANDBY, TO LOGICAL 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:

```
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.

On the target logical standby database:

Step 3 Prepare the target logical standby database for the switchover.

Use the following statement to build a LogMiner 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 log 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, this operation can take some time to complete.

On the current primary database:

Step 4 Verify the switchover status in the V$DATABASE view.

Before you transition the primary database to the logical standby role, verify the LogMiner dictionary was received by the primary database by querying the SWITCHOVER_STATUS column of the V$DATABASE fixed view on the 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
```

Step 5 Switch the primary database to the logical standby database role.

To transition the primary database to a logical standby database role, 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. It also puts a marker in the redo data to provide a synchronization point for logical standby database operations. 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 is now transitioned to run in the standby database role.

When you transition a primary database to a logical standby database role, you do not have to shut down and restart the database.

On the target logical standby database (new primary database):

Step 6 Verify the switchover status in the V$DATABASE view.

After you transition 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. 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 Ensure all standby databases begin receiving redo data.

On the new primary database, perform a log switch to ensure all logical standby databases begin receiving redo data from the new primary database:

```
SQL> ALTER SYSTEM ARCHIVE LOG CURRENT;
```

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.

During failovers involving a logical standby database:

- In all cases, the original primary database and all physical standby databases are no longer compatible and cannot remain in the new Data Guard configuration.

- In most cases, other logical 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.

Before starting the failover, perform as many of the steps documented in Section 7.1.3.1 as possible to prepare the selected standby database for the failover. Depending on the protection mode for the configuration and the attributes you chose for log 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 3. Otherwise, begin with Step 1 to determine if any manual gap resolution steps must be performed.

On the logical standby database being transitioned to the primary role:

Step 1 Copy and register any missing archived redo log files.

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 the missing log files from the primary database to the logical standby database.
3. Register the copied log files.

On the logical standby database, query the DBA_LOGSTDBY_LOG view to determine which log files are missing and then register them. 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 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_1.arc
         1         10 /disk1/oracle/dbs/log-1292880008_10_1.arc
```

To resolve the gap, copy the missing archived redo log files for `THREAD 1` (with sequence numbers 7, 8, and 9). Then, register these archived redo log files on the logical standby database. For example:

```
SQL> ALTER DATABASE REGISTER LOGICAL LOGFILE
  2> '/disk1/oracle/dbs/log-%r_%s_%t.arc';
Database altered.
```

After you copy and register the missing archived redo log files to the logical standby system, query the `DBA_LOGSTDBY_LOG` view again to ensure there are no more gaps and the next thread and sequence number needed by the target logical standby database do not exist.

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 `DBA_LOGSTDBY_PROGRESS` view. For example:

```
SQL> SELECT APPLIED_SCN, NEWEST_SCN FROM DBA_LOGSTDBY_PROGRESS;

APPLIED_SCN NEWEST_SCN
----------- ----------
     190725     190725
```

When the `APPLIED_SCN` and `NEWEST_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 NODELAY FINISH;
Database altered.
```

See Chapter 9 and Chapter 10 for information about the DBA_LOGSTDBY_PROGRESS view.

Step 3 Enable remote destinations.

If you have not previously configured role-based destinations as described in Section 7.1.3.1, 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 and Chapter 12 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 statements on the target logical standby database (that you are transitioning to the new primary role) to stop SQL Apply and activate the database in the primary database role:

```
SQL> ALTER DATABASE STOP LOGICAL STANDBY APPLY;
SQL> ALTER DATABASE ACTIVATE LOGICAL STANDBY DATABASE;
```

On all other logical standby databases:

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 DATABASE 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.

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 by issuing this SQL statement on each logical standby 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
```

On the new primary database:

Step 7 Optionally, back up the new primary database.

Optionally, perform a closed backup of the new primary database. In place of a closed backup, instead consider performing an open backup of the database. Immediately performing a backup, while not required, is a recommended safety measure, because you cannot recover changes made after the failover without a complete backup copy of the database.

Step 8 Optionally, restore the failed primary database.

After performing a failover, you can optionally restore the failed primary database as a new standby database using either 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 10.3, "Using Flashback Database After a Failover".

  Note: You must have already enabled Flashback Database on the old primary database before the failover. See *Oracle Database Backup and Recovery Advanced User's Guide* for more information.

- Re-create the failed database and add it to the configuration as a new standby database following the procedure in Section 3.2, "Creating a Physical Standby Database"or Section 4.2, "Creating a Logical Standby Database".

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.

8

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
- Using a Standby Database That Is Open for Read-Only Access
- Managing Primary Database Events That Affect the Standby Database
- Using RMAN to Back Up and Restore Files on a Physical Standby Database
- Recovering Through the OPEN RESETLOGS Statement
- Monitoring the Primary and Standby Databases

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 opening the database for read-only access. Typically, you start Redo Apply. The following example shows how to start a physical standby database:

  1. Start and mount the database:

     ```
     SQL> STARTUP MOUNT;
     ```

  2. Start log apply services:

     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 standby database for read-only access.

Note: When you first start log apply services 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 the MRP is unable to determine the starting sequence number for media recovery. If this occurs, you must either manually retrieve and register the archived redo log files on the standby database, or wait for the automatic archiving to occur before restarting log apply services.

8.1.2 Shutting Down a Physical Standby Database

To shut down a physical standby database and stop log apply services, use the SQL*Plus `SHUTDOWN IMMEDIATE` 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 log apply services 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 log apply services are running, cancel them as shown in the following example:

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

3. Shut down the standby database.

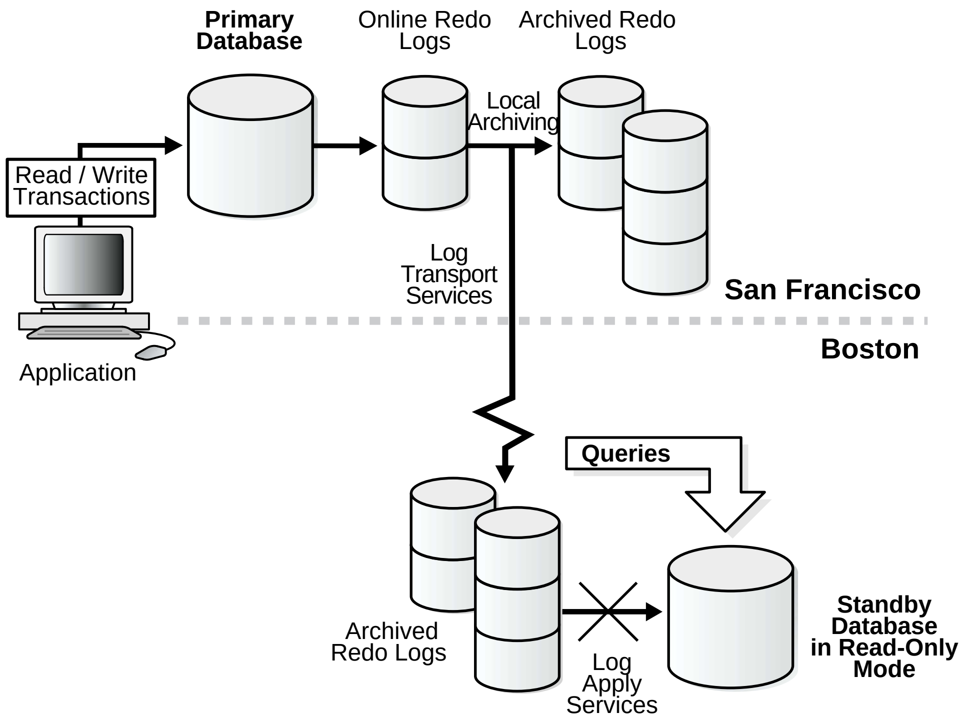
   ```
   SQL> SHUTDOWN;
   ```

8.2 Using a Standby Database That Is Open for Read-Only 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 log apply services are 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



This section contains the following topics:

- Assessing Whether or Not to Open a Standby Database for Read-Only Access
- Opening a Physical Standby Database for Read-Only Access
- Sorting Considerations for Standby Databases Open for Read-Only Access

8.2.1 Assessing Whether or Not to Open a Standby Database for Read-Only Access

As you decide whether or not to open a physical standby database for read-only access, consider the following:

- When a physical standby database is open for read-only access, redo data from the primary database is received by the standby database, but the log files are not applied. Therefore, a standby database that is open for read-only access is not current with the primary database. 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 to use a standby system for reporting purposes and also maintain 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 and the system resources available on the standby system, you might:
  - Configure two physical standby databases on the standby system 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 on the standby system to maintain a block-for-block 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 or real-time apply:

1. Cancel Redo Apply or real-time 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 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;
   ```

You do not need to shut down the instance to start either of these apply modes.

8.2.3 Sorting Considerations for Standby Databases Open for Read-Only Access

Before you open your standby database for read-only access, consider the following topics regarding sorting operations:

- Sorting Operations While the Database Is Open for Read-Only Access
- Sorting Operations Without Temporary Tablespaces

8.2.3.1 Sorting Operations While the Database Is Open for Read-Only Access

To perform queries that sort a large amount of data on a standby database that is open for read-only access, the Oracle database must be able to perform on-disk sorting. You cannot allocate space for sorting in tablespaces that cause Oracle software to write to the data dictionary.

Temporary tablespaces allow you to add `tempfile` entries when the database is open for read-only access for the purpose of making queries without affecting dictionary files or generating redo entries. Therefore, you can use temporary tablespaces as long as you follow these requirements for creating them:

- The tablespaces must be temporary, locally managed, and contain only temporary files.
- User-level allocations and permissions to use the locally managed temporary tablespaces must be in place on the primary database. You cannot change these settings on the standby database.
- You must create and associate a temporary file for the temporary tablespace on the standby database.

To create a temporary tablespace for use on a read-only physical standby database:

If you did not have a temporary tablespace on the primary database when you created the physical standby database, perform the following steps on the primary database:

1. Enter the following SQL statement:

```
SQL> CREATE TEMPORARY TABLESPACE temp1
     TEMPFILE '/disk1/oracle/oradata/payroll/temp1.dbf'
     SIZE 20M REUSE
     EXTENT MANAGEMENT LOCAL UNIFORM SIZE 16M;
```

2. Switch the log file to send the redo data to the standby database:

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

To create and associate a temporary file with a temporary tablespace on a read-only physical standby database:

The redo data that is generated on the primary database automatically creates the temporary tablespace in the standby control file after the archived redo log file is applied to the physical standby database. However, even if the temporary tablespace existed on the primary database before you created the physical standby database, you must use the `ADD TEMPFILE` clause to actually create the disk file on the standby database.

On the physical standby database, perform the following steps:

1. Start Redo Apply or real-time apply, if necessary, and apply the archived redo log files. To start Redo Apply, issue the following SQL*Plus 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;
   ```

2. Connect to the standby database and query the `V$ARCHIVED_LOG` view to verify all of the archived redo log files have been 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.
   ```

3. Cancel log apply services and open the physical standby database for read-only access using the following SQL statements:

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

   Opening the physical standby database for read-only access enables you to add a temporary file. Because adding a temporary file does not generate redo data, it is allowed for a database that is open for read-only access.

4. Create a temporary file for the temporary tablespace. The size and names for the files can differ from the primary database. For example:

   ```
   SQL> ALTER TABLESPACE temp1
          ADD TEMPFILE '/disk1/oracle/oradata/payroll/s_temp1.dbf'
          SIZE 10M REUSE;
   ```

8.2.3.2 Sorting Operations Without Temporary Tablespaces

If a temporary file does not exist on the standby database, or if the standby database is not open and you attempt to sort a large amount of data, an error is returned, as shown in the following example.

```
SQL> SELECT * FROM V$PARAMETER;

select * from v$parameter

              *

ERROR at line 1:

ORA-01220: file based sort illegal before database is open
```

Note that you can, however, sort small amounts of data if the SORT_AREA_SIZE parameter is set to a sufficient value in your server parameter file. (The SORT_AREA_SIZE parameter is a static parameter. See *Oracle Database Reference* for information about setting this initialization parameter.)

8.3 Managing Primary Database Events That Affect the Standby Database

To prevent possible problems, you must be aware of events in 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 archived redo log files 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 log transport services and log apply services, 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 STANDBY_FILE_MANAGEMENT initialization parameter to AUTO, 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 DROP or DELETE 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	Alter the primary database control file (using the SQL ALTER DATABASE CREATE CONTROLFILE statement)	Re-create the standby control file or re-create the standby database, depending on the alteration made.
Section 8.3.7	Perform a DML or DDL operation using the NOLOGGING or UNRECOVERABLE clause	Send the datafile containing the unlogged changes to the standby database.
Chapter 11	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 Adding a Tablespace and a Datafile 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 Adding a Tablespace and a Datafile When STANDBY_FILE_ MANAGEMENT Is Set to MANUAL

The following example shows the steps required 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.

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. 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
```

3. Perform the following steps to copy the tablespace to a remote standby location:

   a. Place the new tablespace offline:

   ```
   SQL> ALTER TABLESPACE new_ts OFFLINE;
   ```

   b. Copy the new tablespace to a local temporary location using an operating system utility copy command. Copying the files to a temporary location will reduce the amount of time the tablespace must remain offline. The following example copies the tablespace using the UNIX cp command:

   ```
   % cp /disk1/oracle/oradata/payroll/t_db2.dbf
   /disk1/oracle/oradata/payroll/s2t_db2.dbf
   ```

   c. Place the new tablespace back online:

   ```
   SQL> ALTER TABLESPACE new_ts ONLINE;
   ```

   d. Copy the local copy of the tablespace to a remote standby location using an operating system utility command. The following example uses the UNIX rcp command:

   ```
   %rcp /disk1/oracle/oradata/payroll/s2t_db2.dbf standby_location
   ```

4. Archive the current online redo log file on the primary database so it will get transmitted to the standby database:

   ```
   SQL> ALTER SYSTEM ARCHIVE LOG CURRENT;
   ```

5. Use the following query to make sure that Redo Apply is running. If the MRP or MRP0 process is returned, Redo Apply is being performed.

   ```
   SQL> SELECT PROCESS, STATUS FROM V$MANAGED_STANDBY;
   ```

6. Verify the datafile was added to the standby database after the archived redo log file was applied 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
   ```

8.3.2 Dropping a Tablespace in the Primary Database

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 in the standby database. The following sections provide examples of dropping a datafile to the primary and standby databases when the STANDBY_FILE_MANAGEMENT initialization parameter is set to AUTO and MANUAL. To verify any deleted datafiles are no longer part of the database, query the V$DATAFILE view.

8.3.2.1 Dropping a Tablespace and a Datafile 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 at the primary site:

   ```
   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;
   ```

Optionally, you can query the V$DATAFILE view to verify any deleted datafiles are no longer part of the database.

3. Delete the corresponding datafile on the standby site 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 Dropping a Tablespace and a Datafile When STANDBY_FILE_MANAGEMENT Is Set to AUTO

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 NCLUDING 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 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 to the 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 was applied and 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. If you do not want the standby database to have the same physical structure as the primary database, then these steps are not required.

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 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 must add a corresponding standby redo log file 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 Altering the Primary Database Control File

Using the SQL `CREATE CONTROLFILE` statement with the `RESETLOGS` option on your primary database will force the primary database to reset the online redo log file the next time the primary database is opened, thereby invalidating the standby database.

If you invalidated the control file for the standby database, re-create the file using the procedure provided in Section 4.2.3.3.

If you invalidated the standby database, you must re-create the standby database using the procedures in Chapter 4.

8.3.7 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.

If you perform an unrecoverable operation (such as a direct path load), you will see a performance improvement on the primary database; but there is no corresponding recovery process performance improvement on the standby database, and you will have to move the data manually to the standby database.

See Section 10.7 for information about recovering after the `NOLOGGING` clause is used.

8.4 Using RMAN to Back Up and Restore Files on a Physical Standby Database

This section describes backup strategies using Oracle Recovery Manager utility (RMAN) with Data Guard and physical standby databases. RMAN is an easy-to-use tool that can take 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.

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.

8.4.1 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 impact of backup operations 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 flash recovery area on an on-demand basis. The flash recovery area acts as disk cache for tape backups.

8.4.1.1 Using Disk as Cache for Tape Backup

This section assumes the flash recovery area is configured (see Section 5.2.3) and other RMAN persistent configurations are set. On the primary database, use 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 8.4.2.4).

On the physical standby database, use the following commands every day to roll forward a level 0 copy of the database. These commands apply the level 1 incremental taken a day before, create a new level 1 incremental, back up archived redo log files to the flash recovery area, and back up files created by the standby instance from flash recovery area to tape:

```
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;
```

8.4.1.2 Performing Backups Directly to Tape

If all backups are written directly to tape, configure the default device type to SBT using the RMAN command `CONFIGURE DEFAULT DEVICE TYPE TO SBT.`

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 8.4.2.4).

Assuming that a complete database backup is taken every Sunday, the following commands can be executed 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;
```

8.4.2 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 change media configuration, or specify different PARMS 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.

8.4.2.1 Recovery from Loss of Datafiles on the Primary Database

Execute 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...>;
```

Execute 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, ...>
```

8.4.2.2 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, 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, as follows:

   ```
   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. Execute 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.

8.4.2.3 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 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:

```
STARTUP MOUNT;
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, as described in Section 8.4.2.

8.4.2.4 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 8.4.2.3, 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 SQL `ALTER DATABASE BACKUP CONTROLFILE TO TRACE NORESETLOGS` statement. 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.

It should be noted that 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 done 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, use 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 10.1.0.1, all the backups 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.

8.4.2.5 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, use the SQL ALTER DATABASE ADD LOGFILE MEMBER '*log_file_name*' REUSE TO GROUP *n* statement. You can do this even when the database is open, without affecting database availability.

If all the 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 procedure.

8.4.2.6 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 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 10.4 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
   ```

   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 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.

2. 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 reset logs branch, and opens the database. See Section 8.5 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, execute 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.

8.4.3 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.

8.4.3.1 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 8.4.2, 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.

8.4.3.2 Standby Database Does Not Contain Datafiles, Used as a Fetch Archived Log (FAL) Server

Use the same procedure described in Section 8.4.1, 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.

8.4.3.3 Standby Database File Names Are Different than Primary Database

If the database filenames are not the same on primary and standby database, 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 the standby database creation.

8.4.4 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 *all mandatory standby databases*.
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 *all mandatory cascading standby databases*.
A standby database that has no mandatory cascading standby databases	Archived redo log files in the flash recovery area that were applied on *the standby database*.

See Appendix C for more information about cascaded redo log 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;
   ```

8.4.4.1 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.

8.4.4.2 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;
```

8.5 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 stops and the managed recovery process (MRP) on the standby database terminates. At this point, you can resynchronize the standby database with the primary database branch, as described in the following table:

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)	Restarting media recovery will automatically recover the standby database into the new branch.	Restart Redo Apply to continue applying redo data. 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 *in the future* of the new branch of redo data.	**1.** Follow the procedure in Section 10.4.1 to flash back a physical standby database. **2.** Restart Redo Apply to continue application of redo data onto new reset logs branch. The MRP automatically resynchronizes the standby database with the new branch.
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.

If the standby database. . .	Then. . .	Perform these steps. . .
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.

8.6 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

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 ALTER DATABASE statement is issued with the ENABLE THREAD or DISABLE THREAD clause specified	■ Alert log ■ V$THREAD view	Alert log
Redo log changed	■ Alert log ■ V$LOG view ■ STATUS column of V$LOGFILE view	Alert log
CREATE CONTROLFILE statement issued	Alert log	Alert log **Note:** When you issue a CREATE CONTROLFILE statement on the primary database, the standby database functions normally until it encounters redo data that depends on initialization parameters.
Managed recovery performed	Alert log	Alert log
Tablespace status changes made (made read / write or read-only, placed online or offline)	■ DBA_TABLESPACES view ■ Alert log	V$RECOVER_FILE view
Datafile added or tablespace created	■ DBA_DATA_FILES view ■ Alert log	V$DATAFILE view Alert log
Tablespace dropped	■ DBA_DATA_FILES view ■ Alert log	V$DATAFILE view Alert log
Tablespace or datafile taken offline, or datafile is deleted offline	■ V$RECOVER_FILE view ■ Alert log	V$RECOVER_FILE view
Rename datafile	■ V$DATAFILE ■ Alert log	V$DATAFILE view Alert log
Unlogged or unrecoverable operations	■ V$DATAFILE view ■ V$DATABASE view	Alert log
Recovery progress	■ V$ARCHIVE_DEST_STATUS view ■ Alert log	V$ARCHIVED_LOG view V$LOG_HISTORY view V$MANAGED_STANDBY view Alert log

Table 8–2 (Cont.) Location Where Common Actions on the Primary Database Can Be Monitored

Primary Database Event	Primary Site Information	Standby Site Information
Log transport status and progress	■ `V$ARCHIVE_DEST_STATUS view` ■ `V$ARCHIVED_LOG view` ■ `V$ARCHIVE_DEST view` ■ Alert log	`V$ARCHIVED_LOG view` Alert log
Autoextend a datafile	Alert log	Alert log
Issue `OPEN RESETLOGS` or `CLEAR UNARCHIVED LOGFILES` statements	Alert log	Alert log
Change initialization parameter	Alert log	Alert log

8.6.1 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.

8.6.2 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. 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 14, "Views Relevant to Oracle Data Guard" and *Oracle Database Reference* for additional information.

8.6.3 Monitoring Recovery Progress

This section shows some samples of the types of views discussed in Section 8.6.2 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 File
- Viewing Database Incarnations Before and After an OPEN RESETLOGS Operation
- 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

8.6.3.1 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 log transport and log 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
```

8.6.3.2 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
```

8.6.3.3 Determining the Location and Creator of the Archived Redo Log File

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.
```

8.6.3.4 Viewing Database Incarnations Before and After an OPEN RESETLOGS Operation

Query the V$DATABASE_INCARNATION view on the standby database to monitor database incarnations and RESETLOGS IDs.

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,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,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.
```

8.6.3.5 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.
```

8.6.3.6 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 file was applied on the standby database. The column displays YES for the log file that was applied. 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.
```

8.6.3.7 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.

9 Managing a Logical Standby Database

This chapter describes how to manage logical standby databases. This chapter contains the following topics:

- Configuring and Managing a Logical Standby Database
- Upgrading the Oracle Database Software Version
- Recovering Through the OPEN RESETLOGS Statement
- Tuning Logical Standby Databases

The topics in this chapter describe how to use SQL statements, initialization parameters, views, and the `DBMS_LOGSTDBY` PL/SQL package to manage logical standby databases.

See *Oracle Data Guard Broker* to use the Data Guard broker to automate the management tasks described in this chapter.

9.1 Configuring and Managing a Logical Standby Database

The `DBMS_LOGSTDBY` PL/SQL package provides procedures to help you configure and manage a logical standby database. You can use the `DBMS_LOGSTDBY` PL/SQL package to perform management tasks such as the following on a logical standby database:

- Managing SQL Apply
- Controlling User Access to Tables in a Logical Standby Database
- Deleting Archived Redo Log Files No Longer Needed By SQL Apply
- Modifying a Logical Standby Database
- How Triggers and Constraints Are Handled on a Logical Standby Database

- Skipping SQL Statements on a Logical Standby Database
- Adding or Re-creating Tables on a Logical Standby Database
- Viewing and Controlling Logical Standby Events
- Understanding and Viewing SQL Apply Activity
- Enabling Real-Time Apply
- Determining How Much Redo Data Was Applied
- Recovering from Errors
- Refreshing Materialized Views

Note: Users requiring access to the `DBMS_LOGSTDBY` package must be granted the `LOGSTDBY_ADMINISTRATOR` role.

9.1.1 Managing SQL Apply

The `DBMS_LOGSTDBY` PL/SQL package includes procedures to help you manage SQL Apply on a logical standby database. Using it you can do the following:

- Provide a way to skip applying archived redo log files or standby redo log files to selected tables or entire schemas in the standby database
- Manage initialization parameters used by SQL Apply
- Ensure supplemental logging is enabled properly
- Describe a set of operations that should not be applied to the logical standby database
- Avoid applying DML or DDL changes for temporary tables
- Avoid applying any `CREATE`, `ALTER`, or `DROP INDEX` operations
- Record the error and continue applying the archived redo log files or standby redo log files to the logical standby database if an error occurs while applying a DDL statement
- Stop SQL Apply and wait for the DBA to specify what action should be taken when an error occurs on a DDL statement

Table 9–1 summarizes the procedures of the `DBMS_LOGSTDBY` PL/SQL package.

Table 9–1 Procedures of the DBMS_LOGSTDBY PL/SQL Package

Subprograms	**Description**
`APPLY_SET`	Enables you to set the values of specific initialization parameters to configure and maintain SQL Apply.
`APPLY_UNSET`	Resets the value of specific initialization parameters to the system default values.
`BUILD`	Ensures supplemental logging is enabled correctly and builds the LogMiner dictionary.
`INSTANTIATE_TABLE`	Creates and populates a table in the standby database from a corresponding table in the primary database.
`SKIP`	Enables you to specify which database operations done on the primary database will not be applied to the logical standby database.
`SKIP_ERROR`	Specifies criteria to follow if an error is encountered. You can stop SQL Apply or ignore the error.
`SKIP_TRANSACTION`	Specifies transaction identification information to skip (ignore) while applying specific transactions to the logical standby database. This subprogram also allows alternate statements to be executed.
`UNSKIP`	Modifies the options set in the `SKIP` procedure.
`UNSKIP_ERROR`	Modifies the options set in the `SKIP_ERROR` procedure.
`UNSKIP_TRANSACTION`	Modifies the options set in the `SKIP_TRANSACTION` procedure.

See *PL/SQL Packages and Types Reference* for complete information about the `DBMS_LOGSTDBY` package.

9.1.2 Controlling User Access to Tables in a Logical Standby Database

The SQL `ALTER DATABASE GUARD` statement controls user access to tables in a logical standby database. The database guard is set to `ALL` by default on a logical standby database.

The `ALTER DATABASE GUARD` statement allows the following keywords:

- `ALL`

  Specify `ALL` to prevent all users, other than `SYS`, from making changes to any data in the logical standby database.

- `STANDBY`

  Specify `STANDBY` to prevent all users, other than `SYS`, from making DML and DDL changes to any table or sequence being maintained through SQL Apply.

- `NONE`

  Specify `NONE` if you want typical security for all data in the database.

For example, use the following statement to enable users to modify tables not maintained by SQL Apply:

```
SQL> ALTER DATABASE GUARD STANDBY;
```

Privileged users can temporarily turn the database guard off and on for the current session using the `ALTER SESSION DISABLE GUARD` and `ALTER SESSION ENABLE GUARD` statements, respectively. This statement replaces the `DBMS_LOGSTDBY.GUARD_BYPASS` PL/SQL procedure that performed the same function in Oracle9*i*. The `ALTER SESSION [ENABLE|DISABLE] GUARD` statement is useful when you want to temporarily disable the database guard to make changes to the database, as described in Section 9.1.4.

Note: Be careful not to let the primary and logical standby databases diverge while the database guard is disabled.

9.1.3 Deleting Archived Redo Log Files No Longer Needed By SQL Apply

Periodically, you need to remove archived redo log files that are no longer needed by SQL Apply to reclaim disk space. Perform the following steps to remove archived redo log files from the file system:

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.

9.1.4 Modifying a Logical Standby 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.

9.1.4.1 Performing DDL on a Logical Standby Database

This section describes how to add an index 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 reenable 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.
```

This sample procedure could be used to execute other DDL statements. Oracle recommends that you do not perform DML operations 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. It is unlikely that you will be able to modify rows in a table in such a way that the logical standby database can incrementally maintain the rows.

9.1.4.2 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 an 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 `MYSCHEMA`. They will no longer be maintained through SQL Apply.

```
SQL> ALTER DATABASE STOP LOGICAL STANDBY APPLY;
Database altered.

SQL> EXECUTE DBMS_LOGSTDBY.SKIP('SCHEMA_DDL','MYSCHEMA','MYTABLES%');
PL/SQL procedure successfully completed.

SQL> EXECUTE DBMS_LOGSTDBY.SKIP('DML','MYSCHEMA','MYTABLES%');
PL/SQL procedure successfully completed.
```

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY;
Database altered.
```

The example then queries the `DBA_LOGSTDBY_PARAMETERS` view to verify the logical standby database is updated. Verification can take a while so you might need to repeat the query until no rows are returned, as shown in the following example:

```
SQL> SELECT VALUE FROM DBA_LOGSTDBY_PARAMETERS WHERE NAME = 'GUARD_STANDBY';
VALUE
---------
Ready
```

Finally, the example sets the database guard to allow updates to the tables.

```
SQL> ALTER DATABASE GUARD STANDBY;
Database altered.
```

9.1.5 How Triggers and Constraints Are Handled on a Logical Standby Database

You do not need to take any action to enable or handle triggers and constraints on logical standby databases. Triggers and constraints are enabled on the standby database but they are not executed. The following describes how triggers and constraints are handled on a logical standby database:

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.

9.1.6 Skipping SQL Statements on a Logical Standby Database

If only a subset of activity on a primary database is of interest on the standby database, use the `DBMS_LOGSTDBY.SKIP` procedure to define filters that prevent SQL Apply from issuing the SQL statements on the logical standby database. (See

Section 4.1.1.1 for information about SQL statements that are skipped automatically.)

Tables continue applying SQL statements after filtering out unsupported datatypes or statements automatically. However, you must use the DBMS_LOGSTDBY.SKIP procedure to skip tables that you do not want to apply to the logical standby database. The following list shows typical examples of the types of SQL statements that can be filtered or skipped so that they are not applied on the logical standby database:

- DML or DDL changes for tables
- CREATE, ALTER, or DROP INDEX DDL statements
- CREATE, ALTER, DROP, or TRUNCATE TABLE statements
- CREATE, ALTER, or DROP TABLESPACE statements
- CREATE or DROP VIEW statements

Example 9–1 demonstrates how to skip all SQL statements that reference the EMP table in a logical standby database.

Example 9–1 Skipping a Table in a Logical Standby Database

```
SQL> ALTER DATABASE STOP LOGICAL STANDBY APPLY;
SQL> EXECUTE DBMS_LOGSTDBY.SKIP('SCHEMA_DDL', 'SCOTT', 'EMP', NULL);
SQL> EXECUTE DBMS_LOGSTDBY.SKIP('DML', 'SCOTT', 'EMP', NULL);
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY;
```

In addition to skipping DML and DDL statements for schema and non-schema operations, you can also skip specific DML and DDL operations as well. Example 9–2 shows how to skip ALTER TABLESPACE and CREATE TABLESPACE for non-schema DDL operations.

Example 9–2 Skipping ALTER or CREATE TABLESPACE Statements

```
SQL> EXEC DBMS_LOGSTDBY.SKIP('CREATE TABLESPACE', NULL, NULL, NULL);
SQL> EXEC DBMS_LOGSTDBY.SKIP('ALTER TABLESPACE', NULL, NULL, NULL);

SQL> COLUMN ERROR FORMAT a5;
SQL> COLUMN STATEMENT_OPT FORMAT a20;
SQL> COLUMN OWNER FORMAT a10
SQL> COLUMN NAME FORMAT a15;
SQL> COLUMN PROC FORMAT a20;
SQL> SELECT * FROM DBA_LOGSTDBY_SKIP;
```

```
ERROR STATEMENT_OPT     OWNER      NAME            PROC
----- ----------------- ---------- --------------- --------------------
N     CREATE TABLESPACE
N     ALTER TABLESPACE
```

9.1.7 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 and Section 4.2.2.1 that explain:

- How to ensure table rows in the primary database can be uniquely identified
- How to determine if the primary database contains datatypes or tables that are not supported by a logical standby database

The following list and Example 9–3 show how to re-create a table and resume SQL Apply on that table:

1. Stop SQL Apply.
2. Ensure no operations are being skipped for that table by querying the DBA_LOGSTDBY_SKIP view.

   If any operations are being skipped for that table, resume application of each operation that is currently being skipped by using the DBMS_LOGSTDBY.UNSKIP procedure. If multiple filters were created on the table, you will need to execute the procedure multiple times.
3. Re-create the table in the logical standby database using the DBMS_LOGSTDBY.INSTANTIATE_TABLE procedure. In addition to creating a table, this procedure also imports the data from the primary table using a database link.
4. Resume SQL Apply.

Before accessing data in the newly added table, archive the current online redo log file on the primary database and ensure the archived redo log file is applied to the logical standby database.

Example 9–3 demonstrates how to add the EMP table to a logical standby database.

Example 9–3 Adding a Table to a Logical Standby Database

```
SQL> ALTER DATABASE STOP LOGICAL STANDBY APPLY;
SQL> SELECT * FROM DBA_LOGSTDBY_SKIP;

ERROR  STATEMENT_OPT         OWNER          NAME            PROC
---------------------------------------------------------------------
N      SCHEMA_DDL            SCOTT          EMP
N      DML                   SCOTT          EMP

SQL> EXECUTE DBMS_LOGSTDBY.UNSKIP('DML','SCOTT','EMP');
SQL> EXECUTE DBMS_LOGSTDBY.UNSKIP('SCHEMA_DDL','SCOTT','EMP');
SQL> EXECUTE DBMS_LOGSTDBY.INSTANTIATE_TABLE('SCOTT','EMP','DBLINK');
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY;
```

Log on to the primary database and issue the following statements:

```
SQL> ALTER SYSTEM ARCHIVE LOG CURRENT;
SQL> SELECT FIRST_CHANGE# FROM V$LOG WHERE STATUS = 'CURRENT';
```

When the value returned by the DBA_LOGSTDBY_PROGRESS.APPLIED_SCN procedure is equal to (FIRST_CHANGE# - 1) or less than the value selected from the query of the V$LOG view, the database is consistent and you can safely run reports again.

9.1.8 Viewing and Controlling Logical Standby Events

When you query the DBA_LOGSTDBY_EVENTS view, it displays a table of events that records activity about SQL Apply. In particular, DDL execution or anything that generates an error is recorded in the events table. You can control what and how much activity is recorded in the events table. By default, 100 records are stored in this table, but you can increase it. For example:

```
SQL> DBMS_LOGSTDBY.APPLY_SET('MAX_EVENTS_RECORDED', 200);
```

Additionally, you can indicate what type of events you want recorded. By default, everything is recorded in the table. However, you can set the RECORD_SKIP_DDL, RECORD_SKIP_ERRORS, and RECORD_APPLIED_DDL parameters to FALSE to avoid recording these events.

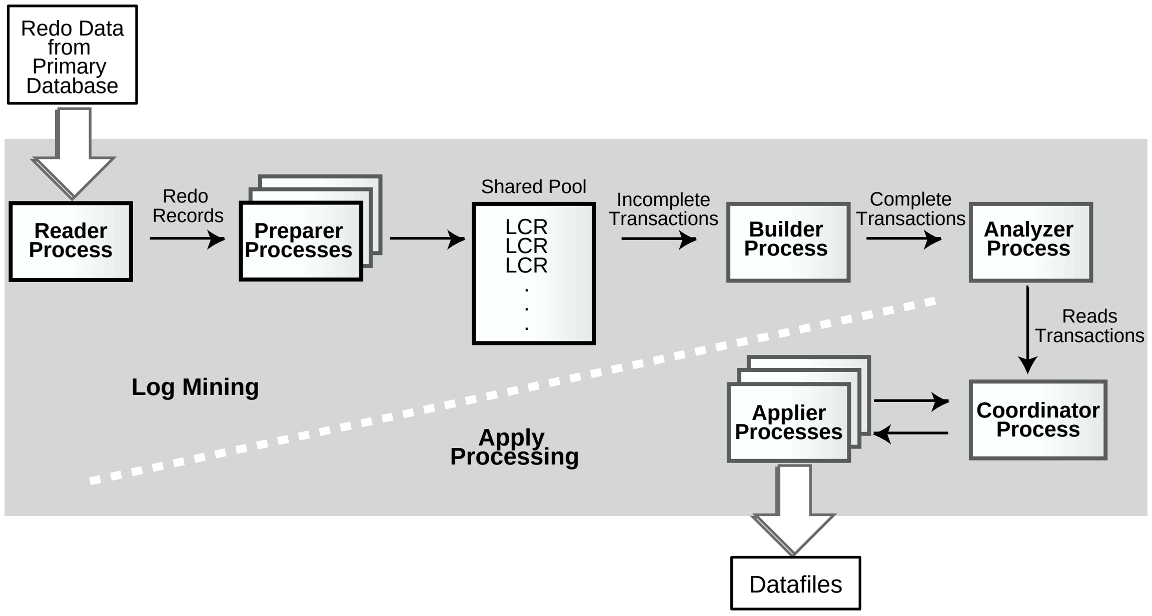
Errors that cause SQL Apply to stop are always recorded in the events table (unless there is insufficient space in the system tablespace). These events are always put into the ALERT.LOG file as well, with the keyword 'LOGSTDBY' included in the text. When querying the view, select the columns in order by EVENT_TIME, COMMIT_

SCN, and CURRENT_SCN. This ordering ensures a shutdown failure appears last in the view.

9.1.9 Understanding and Viewing SQL Apply Activity

SQL Apply uses a collection of parallel execution servers and background processes that 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



In Figure 9–1:

- The READER process reads redo records from the archived redo log files.
- The PREPARER processes do the heavy computing required to convert the block changes into table changes, or logical change records (LCR). At this point, the LCRs do not represent any specific transactions.
- The BUILDER process assembles completed transactions from the individual LCRs.

- The ANALYZER process examines the records, possibly eliminating transactions and identifying dependencies between the 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$LOGSTDBY view to see what each process is currently doing; the TYPE column describes the task being performed. When querying the V$LOGSTDBY view, pay special attention to the HIGH_SCN column. This is an activity indicator. As long as it is changing each time you query the V$LOGSTDBY view, progress is being made. The STATUS column gives a text description of the current activity. For example:

```
SQL> COLUMN NAME FORMAT A30
SQL> COLUMN VALUE FORMAT A30
SQL> SELECT NAME, VALUE FROM V$LOGSTDBY_STATS WHERE NAME = 'coordinator state';
NAME                           VALUE
------------------------------ ------------------------------
coordinator state              APPLYING

SQL> COLUMN STATUS FORMAT A50
SQL> COLUMN TYPE FORMAT A12
SQL> SELECT TYPE, HIGH_SCN, STATUS FROM V$LOGSTDBY;
TYPE           HIGH_SCN STATUS
------------ ---------- --------------------------------------------------
COORDINATOR             ORA-16117: processing
READER                  ORA-16127: stalled waiting for additional transactions
                        to be applied

BUILDER          191896 ORA-16116: no work available
PREPARER         191902 ORA-16117: processing
ANALYZER         191820 ORA-16120: dependencies being computed for transaction
                        at SCN 0x0000.0002ed4e
```

```
APPLIER          191209 ORA-16124: transaction 1 16 1598 is waiting on another
                               transaction
  .
  .
  .
```

Another place to get information about current activity is the `V$LOGSTDBY_STATS` view, which provides state and status information. All of the options for the `DBMS_LOGSTDBY.APPLY_SET` procedure have default values, and those values (default or set) can be seen in the `V$LOGSTDBY_STATS` view. In addition, a count of the number of transactions applied or transactions ready will tell you if transactions are being applied as fast as they are being read. Other statistics include information on all parts of the system. For example:

```
SQL> COLUMN NAME FORMAT A35
SQL> COLUMN VALUE FORMAT A35
SQL> SELECT NAME, VALUE FROM V$LOGSTDBY_STATS
  2> WHERE NAME LIKE 'coordinator%' or NAME LIKE 'transactions%';

NAME                                VALUE
----------------------------------- -----------------------------------
coordinator state                   APPLYING
transactions ready                  7821
transactions applied                7802
coordinator uptime                  73
```

This query shows how long SQL Apply has been running and how many transactions have been applied in that time. It also shows how many transactions are available to be applied.

9.1.10 Enabling Real-Time Apply

By default, Data Guard waits for the *full* archived redo log file to arrive on the standby database before recovering 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, which recovers redo data from the standby redo log files as they are being filled up by the remote file server (RFS) process. With real-time apply enabled, SQL Apply recovers redo data from standby redo log files at the same time the log files are being written to, as opposed to when 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;
```

9.1.11 Determining How Much Redo Data Was Applied

Transaction data in the redo stream can span multiple redo log files. For this reason, logical standby databases use an SCN range of redo data, rather than individual archived redo log files to report the progress of SQL Apply.

The DBA_LOGSTDBY_PROGRESS view displays APPLIED_SCN, NEWEST_SCN, and READ_SCN information. The APPLIED_SCN indicates that committed transactions less than or equal to that SCN were applied. The NEWEST_SCN is the maximum SCN to which data could be applied if no more redo data is received. This is usually the MAX(NEXT_CHANGE#)-1 from DBA_LOGSTDBY_LOG when there are no gaps in the list.

Archived redo log files with a NEXT_CHANGE# value that is less than the READ_SCN value are no longer needed. The information in those log files was applied or persistently stored in the database. The time values associated with these SCN values are only estimates based on log times. They are not meant to be accurate times of when those SCN values were written on the primary database.

To see which archived redo log files were applied or were not applied, issue the following query:

```
SQL> ALTER SESSION SET NLS_DATE_FORMAT  = 'DD-MON-YY HH24:MI:SS';
Session altered.

SQL> SELECT SEQUENCE#, FIRST_TIME, APPLIED
  2  FROM DBA_LOGSTDBY_LOG
  3  ORDER BY SEQUENCE#;

 SEQUENCE# FIRST_TIME         APPLIED
---------- ------------------ -------
        24 23-JUL-02 18:19:05 YES
        25 23-JUL-02 18:19:48 YES
        26 23-JUL-02 18:19:51 YES
        27 23-JUL-02 18:19:54 YES
        28 23-JUL-02 18:19:59 YES
        29 23-JUL-02 18:20:03 YES
        30 23-JUL-02 18:20:13 YES
        31 23-JUL-02 18:20:18 YES
        32 23-JUL-02 18:20:21 YES
        33 23-JUL-02 18:32:11 YES
        34 23-JUL-02 18:32:19 CURRENT
        35 23-JUL-02 19:13:20 CURRENT
        36 23-JUL-02 19:13:43 CURRENT
        37 23-JUL-02 19:13:46 CURRENT
        38 23-JUL-02 19:13:50 CURRENT
        39 23-JUL-02 19:13:54 CURRENT
        40 23-JUL-02 19:14:01 CURRENT
        41 23-JUL-02 19:15:11 NO
        42 23-JUL-02 19:15:54 NO

19 rows selected.
```

9.1.12 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. Tables in the SYS schema are never maintained, because only Oracle metadata is maintained in the SYS schema.

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.

9.1.12.1 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. However, if the structure of the file system on the standby system differs from the file system on the primary system, it is likely that an error might result because the `DB_FILE_NAME_CONVERT` will not convert the filenames of one or more sets of datafiles on the primary database to filenames on the standby database for a logical standby database.

If an error was caused by a DDL transaction that contained a file specification that does 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 reenable 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
      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 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 %d in tablespace
```

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;
Database altered.
```

When SQL Apply restarts, the transaction that failed will be re-executed and applied to the logical standby database.

9.1.12.2 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. Thus, multiple tables might be damaged by such an action.

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;
   ```

   From this point on, DML activity for the SCOTT.EMP table will not be 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;
   ```

3. Because the SCOTT.EMP table will contain records as of when the INSTANTIATE_TABLE procedure was performed (in Step 2), it is possible for

the SCOTT.EMP table to contain records for a department not in the SCOTT.DEPT table.

9.1.13 Refreshing Materialized Views

Materialized views refreshed on the primary database are not automatically refreshed separately on a logical standby database. To refresh materialized views on a logical standby database, use the ALTER SESSION DISABLE GUARD and ENABLE GUARD statements. For example:

```
SQL> ALTER SESSION DISABLE GUARD;
SQL> EXECUTE DBMS_MVIEW.REFRESH ( 'BMVIEW', 'F', '',TRUE,FALSE,0,0,0,FALSE);
SQL> ALTER SESSION ENABLE GUARD;
```

See *PL/SQL Packages and Types Reference* for more information about the DBMS_LOGSTDBY package.

If you are using the DBMS_LOGSTDBY.APPLY_SET procedure but you are not using the FULL option (the default) for the TRANSACTION_CONSISTENCY parameter, you should stop SQL Apply before refreshing materialized views on the logical standby database.

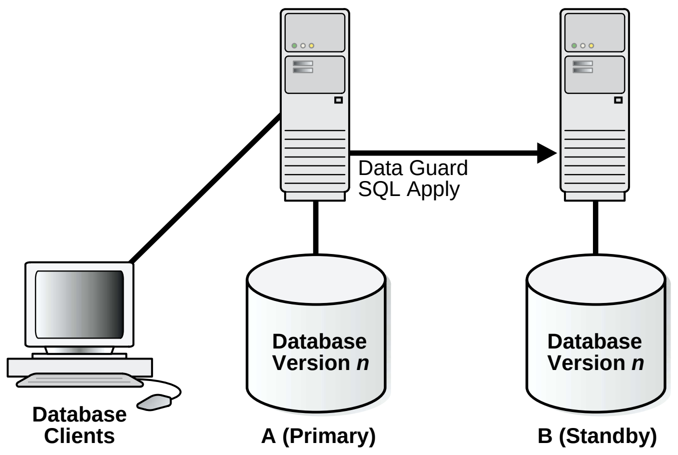
9.2 Upgrading the Oracle Database Software Version

Using a logical standby database, you can upgrade Oracle database software and patch sets with almost no downtime. This section provides a conceptual overview of the upgrade process. For complete database upgrade information, see the ReadMe file for the applicable Oracle Database 10*g* patchset release.

Note: If you cannot use a logical standby database because of the datatypes in your application, then perform the upgrade as documented in *Oracle Database Upgrade Guide*.

Figure 9–2 shows a the Data Guard configuration before the upgrade begins, with the primary and logical standby databases both running the same Oracle software version.

Figure 9–2 Data Guard Configuration Before Upgrade



During the upgrade process:

- The Data Guard configuration operates with mixed database versions at several times so that the upgrade can be validated. The steps in this section indicate where you can end the upgrade and downgrade the software without data loss.
- During these steps, consider having a second standby database in the Data Guard configuration to provide additional data protection.

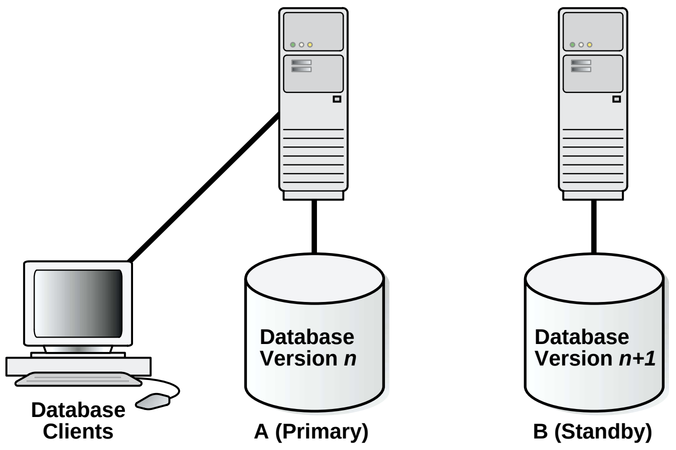
Step 1 Stop SQL Apply and upgrade the logical standby database.

To begin the upgrade, stop SQL Apply and upgrade the Oracle database software on the logical standby database to version *n*+1.

For more information about upgrading the Oracle database software version, see the ReadMe file for the applicable Oracle Database 10*g* patchset release

Figure 9–3 shows the primary database running version *n*, and the logical standby database running version *n*+1. During the upgrade, redo data accumulates on the primary system.

Figure 9–3 Upgrade the Logical Standby Database Version

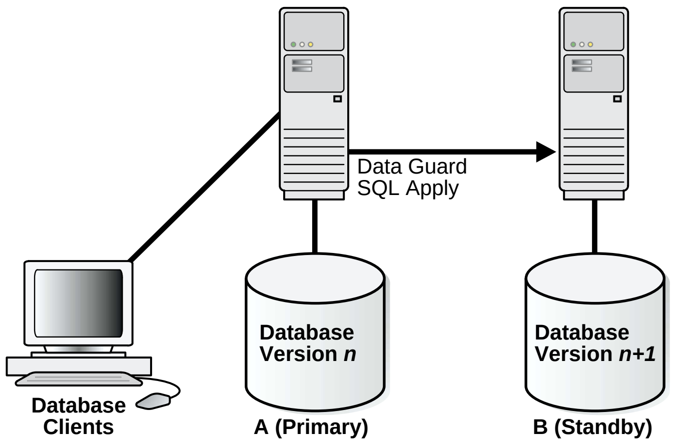


Step 2 Restart SQL Apply.

Restart SQL Apply and operate with version *n* on the primary database and version *n*+1 on the standby database. The Data Guard configuration can run the mixed versions shown in Figure 9–4 for an arbitrary period while you verify the upgraded Oracle software version is running properly in the production environment.

The redo data that was accumulating on the primary system is automatically transmitted and applied on the newly upgraded logical standby database.

Figure 9–4 Running Mixed Versions



Step 3 Perform a switchover.

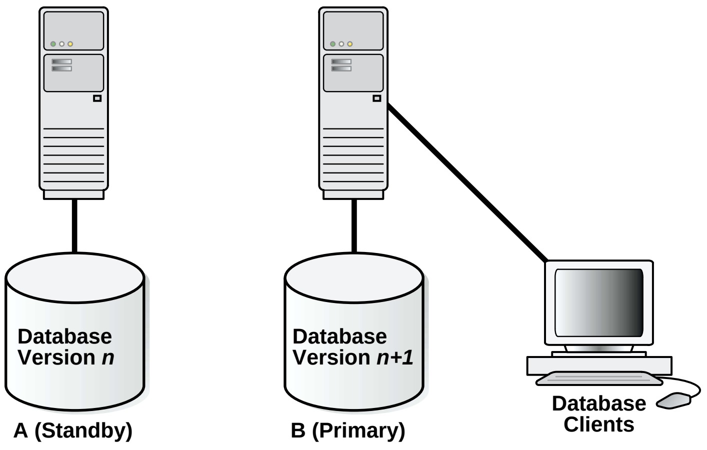
When you are satisfied that the upgraded software is operating properly, you can reverse the database roles by performing a switchover (see Section 7.3.1). This may take only a few seconds. Activate the user applications and services on the new primary database. If application service levels degrade for some reason, then you can open the previous primary database again, switch users back, and quit the previous steps.

After the switchover, you cannot send redo data from the new primary database (B) that is running the new database software version to the new standby database (A) that is running an older software version. This means that:

- Redo data is accumulating on the new primary database.
- The new primary database is unprotected at this time.

Figure 9–5 shows the former standby database (version *n*+1) is now the primary database, and the former primary database (version *n*) is now the standby database. The users are connected to the new primary database.

Figure 9–5 After a Switchover

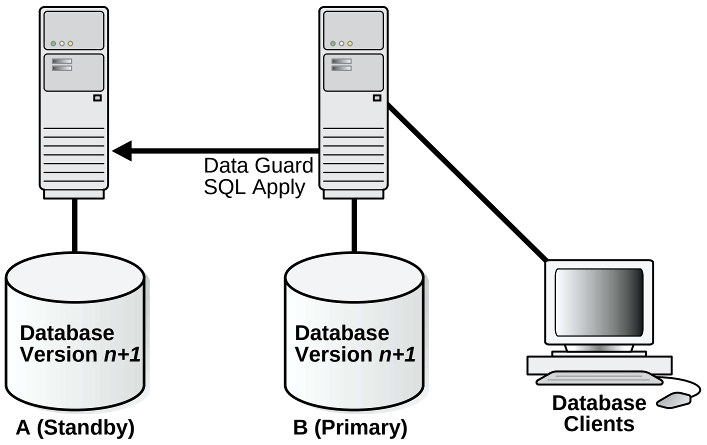


Step 4 Upgrade the new logical standby database.

Upgrade the new logical standby database.

For more information about upgrading the Oracle database software version, see the ReadMe file for the applicable Oracle Database 10*g* patchset release. Figure 9–6 shows the system after both databases were upgraded to version *n*+1.

Figure 9–6 Both Databases Upgraded



Step 5 Start SQL Apply.

When you start SQL Apply, the redo that was accumulating on the primary database is sent to the logical standby database. The primary database is protected against data loss once the redo data is available on the standby database.

Step 6 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 Chapter 11 for more information about the `COMPATIBLE` initialization parameter.

Step 7 Optionally, perform another switchover.

Optionally, perform a another switchover of the databases so the original primary database is once again running in the production database role (as shown in Figure 9–2).

9.3 Recovering Through the OPEN RESETLOGS Statement

Data Guard allows recovery on a logical standby database to continue after the primary database was 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 logical standby database receives a new branch of redo data, SQL Apply stops and the logical standby process (LSP) on the standby database terminates. 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)	No manual intervention is necessary. SQL Apply will automatically take the new branch of redo data.	Restart SQL Apply to continue applying redo data. The LSP automatically resynchronizes the standby database with the new branch of redo data.

If the standby database. . .	Then. . .	Perform these steps. . .
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 *in the future* of the new branch of redo data.	**1.** Follow the procedure in Section 10.4.2 to flash back a logical standby database. **2.** Restart SQL Apply to continue application of redo onto the new reset logs branch. The LSP automatically resynchronizes the standby database with the new branch.
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.
Is missing intervening archived redo log files from the new branch of redo data	The LSP 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 LSP 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.

9.4 Tuning Logical Standby Databases

The following sections describe actions you can take to increase system performance.

9.4.1 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 know the rows are indeed unique because you have ensured this some other way, 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;
```

The following example shows the creation of an index for the EMP table. This should be done for all the tables returned by the previous query:

```
SQL> ALTER SESSION DISABLE GUARD;
SQL> CREATE INDEX EMPI ON EMP (EMPNO);
SQL> ALTER SESSION ENABLE GUARD;
```

See Section 4.1.2 and *Oracle Database SQL Reference* for more information about RELY constraints.

9.4.2 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.

9.4.3 Adjust the Transaction Consistency

Use the TRANSACTION_CONSISTENCY parameter of the DBMS_LOGSTDBY.APPLY_SET procedure to control how transactions are applied to the logical standby database. The default setting is FULL, which applies transactions to the logical standby database in the same order in which they were committed on the primary database.

Regardless of the consistency level chosen, the data in the logical standby database will be transactionally consistent with the primary database when SQL Apply is stopped normally.

Specify one of the following values:

- `FULL` (the default)

  Transactions are applied to the logical standby database in the exact order in which they were committed on the primary database. This option results in the lowest performance, but it is recommended when the logical standby database is used for generic reporting applications.

- `READ_ONLY`

  Transactions are committed out of order, but SQL `SELECT` statements executed on the standby database always return consistent results based on the last consistent SCN known to SQL Apply.

  The `READ_ONLY` option provides better performance than the `FULL` option, and SQL `SELECT` statements return read-consistent results. This is particularly beneficial when you are using the logical standby database to generate reports. The `READ_ONLY` option is recommended when the logical standby database is used for read-only reporting.

  Note: The `READ_ONLY` option should be used only when `ALTER DATABASE GUARD ALL` is set.

- `NONE`

  Transactions are applied out of order from how they were committed on the primary database. This results in the best performance of the three values. The `NONE` option is useful when the logical standby database is in catch-up mode after temporary loss of network connectivity or anytime there a number of logs to apply. This option also works well as long as applications that are reading the logical standby database make no assumptions about transaction order. For example:

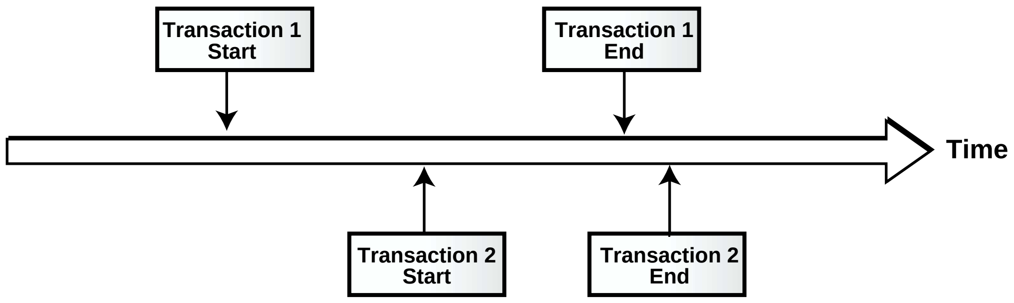
  – On the primary database, one transaction adds a new customer, and a second transaction adds a new order for that customer.

  – On the standby database, those transactions may be reversed. The order for the new customer might be added first. If you then run a reporting application on the standby database that expects to find a customer for the new order, the reporting application might fail because constraints are not checked and triggers are not fired.

For example, the timeline in Figure 9–7 shows Transaction 2 starts a transaction around the time when Transaction 1 ends, and Transaction 2 commits soon after Transaction 1 commits. If `TRANSACTION_CONSISTENCY` is set to:

- `FULL`—SQL Apply guarantees Transaction 1 commits before Transaction 2
- `READ_ONLY`—SQL Apply guarantees either:
  - Transaction 1 will commit before Transaction 2.
  - Transaction 1 and Transaction 2 will commit at the same time.
- `NONE`—SQL Apply does not guarantee order; it is possible that Transaction 2 will commit before Transaction 1.

Figure 9–7 Example of Transaction Consistency with SQL Apply



If you plan to use the logical standby database:

- For reporting or decision support, use the `FULL` or `READ_ONLY` value:
  - Choose `READ_ONLY` if the logical standby database has only one instance
  - Choose `FULL` if the logical standby database has multiple instances (Real Application Clusters)
- For disaster recovery or when SQL Apply needs to catch up, use `NONE`.

See *PL/SQL Packages and Types Reference* for more information about the `DBMS_LOGSTDBY.APPLY_SET` procedure.

9.4.4 Adjust the Maximum Number of Parallel Execution Processes

SQL Apply uses parallel execution processes to perform processing and parallel apply algorithms to maintain a good SQL Apply performance level. You can adjust the maximum number of parallel execution processes for an instance by setting the PARALLEL_MAX_SERVERS initialization parameter. The default value for this parameter is derived from the values of the CPU_COUNT, PARALLEL_AUTOMATIC_TUNING, and PARALLEL_ADAPTIVE_MULTI_USER initialization parameters. This parameter must not be set to a value less than 5 on a logical standby database. However, for best results, set PARALLEL_MAX_SERVERS to a minimum of 9.

You can use the MAX_SERVERS parameter of the DBMS_LOGSTDBY.APPLY_SET procedure to limit the number of parallel servers used by SQL Apply. The default value of this parameter is set to 9. If you set this parameter explicitly, do not set it to a value less than 5, or greater than the value of the PARALLEL_MAX_SERVERS initialization parameter.

Increasing the number of parallel execution processes for an instance can speed up execution operations, but this improvement must be balanced against the consumption of additional system resources by the processes.

9.4.5 Control Memory Usage on the Logical Standby Database

You can use the MAX_SGA parameter of the DBMS_LOGSTDBY.APPLY_SET procedure to set the maximum amount of shared pool space used by SQL Apply for redo cache. By default, SQL Apply will use up to one quarter of the shared pool. Generally speaking, increasing the size of the shared pool or the amount of shared pool space used by SQL Apply will improve the performance of a logical standby database. See *PL/SQL Packages and Types Reference* for more information about the DBMS_LOGSTDBY.APPLY_SET procedure.

10 Data Guard Scenarios

This chapter provides a collection scenarios you might encounter while administering your Data Guard configuration. Each scenario can be adapted to your specific environment. Table 10–1 lists the scenarios presented in this chapter.

Table 10–1 Data Guard Scenarios

Reference	Scenario
Section 10.1	Setting Up and Verifying Archival Destinations
Section 10.2	Choosing the Best Available Standby Database for a Role Transition
Section 10.3	Using Flashback Database After a Failover
Section 10.4	Using Flashback Database After Issuing an Open Resetlogs Statement
Section 10.5	Using a Physical Standby Database with a Time Lag
Section 10.6	Recovering from a Network Failure
Section 10.7	Recovering After the NOLOGGING Clause Is Specified
Section 10.8	Resolving Archive Gaps Manually
Section 10.9	Creating a Standby Database That Uses OMF or ASM

10.1 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:

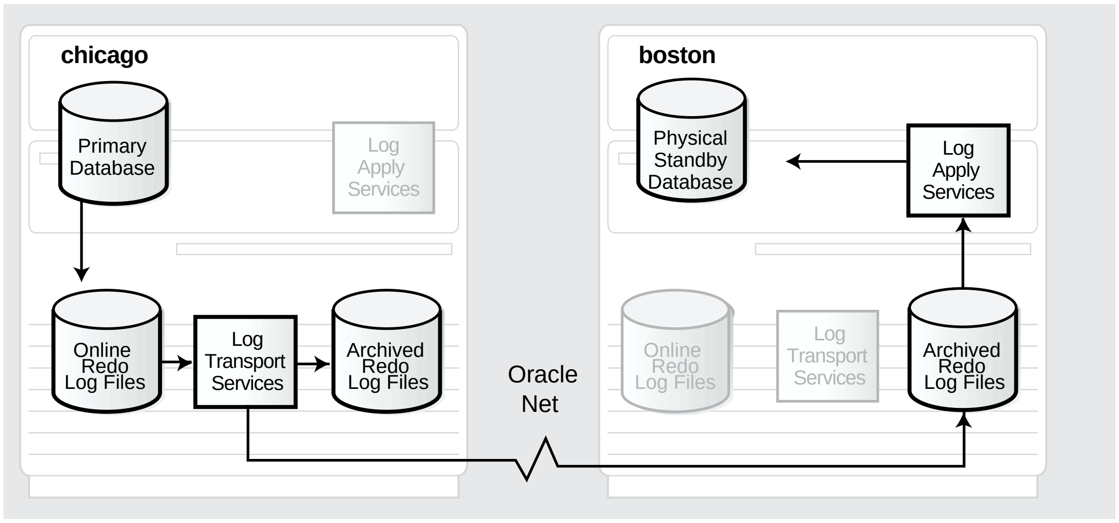
- Configuring a Primary Database and a Physical Standby Database
- Configuring a Primary Database and a Logical Standby Database
- Configuring Both Physical and Logical Standby Databases

- Verifying the Current VALID_FOR Attribute Settings for Each Destination

10.1.1 Configuring a Primary Database and a Physical Standby Database

Figure 10–1 shows the `chicago` primary database, the `boston` physical standby database, and the initialization parameters for each system.

Figure 10–1 Primary and Physical Standby Databases Before a Role Transition



```
DB_UNIQUE_NAME=chicago
LOG_ARCHIVE_CONFIG=
 'DG_CONFIG=(chicago,boston)'
LOG_ARCHIVE_DEST_1=
 'LOCATION=/arch1/chicago/
  VALID_FOR=(ALL_LOGFILES,ALL_ROLES)
  DB_UNIQUE_NAME=chicago'
LOG_ARCHIVE_DEST_2=
 'SERVICE=boston
  VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)
  DB_UNIQUE_NAME=boston'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_STATE_2=ENABLE
STANDBY_ARCHIVE_DEST=/arch1/chicago/
REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE
```

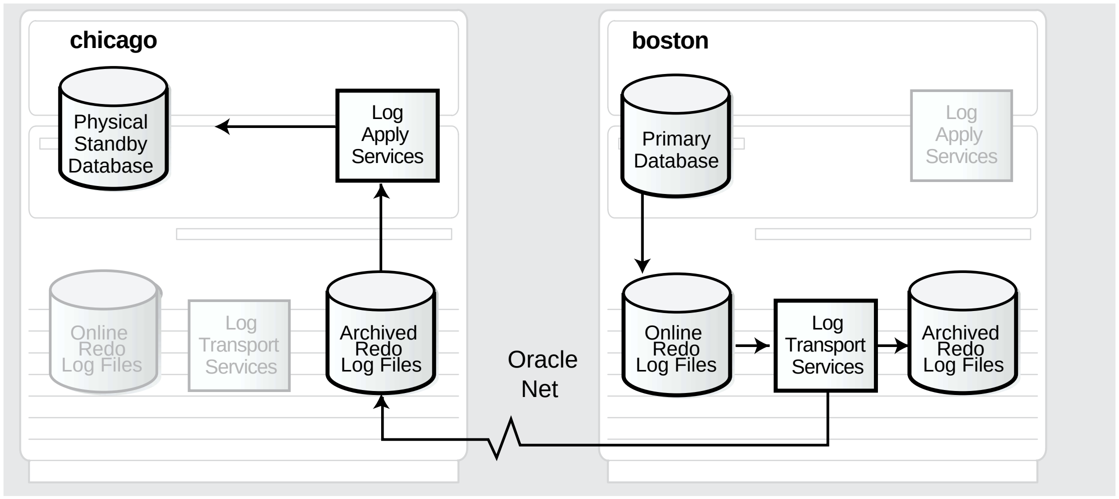
```
DB_UNIQUE_NAME=boston
LOG_ARCHIVE_CONFIG=
 'DG_CONFIG=(chicago,boston)'
LOG_ARCHIVE_DEST_1=
 'LOCATION=/arch1/boston/
  VALID_FOR=(ALL_LOGFILES,ALL_ROLES)
  DB_UNIQUE_NAME=boston'
LOG_ARCHIVE_DEST_2=
 'SERVICE=chicago
  VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)
  DB_UNIQUE_NAME=chicago'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_STATE_2=ENABLE
STANDBY_ARCHIVE_DEST=/arch1/boston/
REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE
```

The following table describes the archival processing shown in Figure 10–1:

	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 `/arch1/boston/` directory.

Figure 10–2 shows the same configuration after a switchover.

Figure 10–2 Primary and Physical Standby Databases After a Role Transition



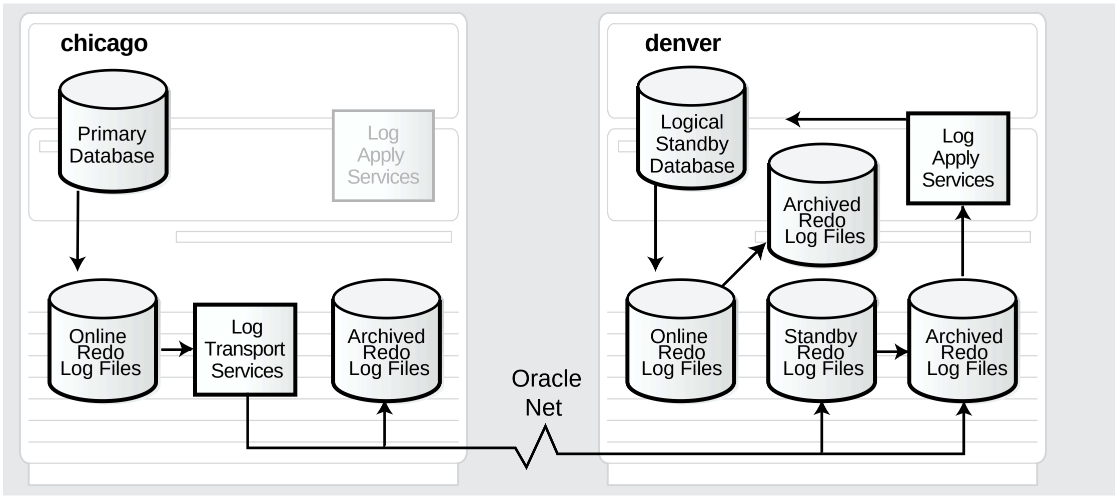
The following table describes the archival processing shown in Figure 10–2:

	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 `chicago` is running in the primary role.	Directs transmission of redo data to the remote physical standby destination `chicago`.
`STANDBY_ARCHIVE_DEST`	Directs archiving of redo data to the archived redo log files in the local `/arch1/chicago/` directory.	Is ignored; valid only when `boston` is running in the standby role.

10.1.2 Configuring a Primary Database and a Logical Standby Database

Figure 10–3 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 10–3 Configuring Destinations for a Primary Database and a Logical Standby Database



```
DB_UNIQUE_NAME=chicago
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_2=
 'LOCATION=/arch2/chicago/
  VALID_FOR=(STANDBY_LOGFILES,STANDBY_ROLE)
  DB_UNIQUE_NAME=chicago'
LOG_ARCHIVE_DEST_3=
 'SERVICE=denver
  VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)
  DB_UNIQUE_NAME=denver'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_STATE_2=ENABLE
LOG_ARCHIVE_DEST_STATE_3=ENABLE
STANDBY_ARCHIVE_DEST=/arch2/chicago/
REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE
```

```
DB_UNIQUE_NAME=denver
LOG_ARCHIVE_CONFIG=
 'DG_CONFIG=(chicago,denver)'
LOG_ARCHIVE_DEST_1=
 'LOCATION=/arch1/denver/
  VALID_FOR=(ONLINE_LOGFILES,ALL_ROLES)
  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_STATE_1=ENABLE
LOG_ARCHIVE_DEST_STATE_2=ENABLE
LOG_ARCHIVE_DEST_STATE_3=ENABLE
STANDBY_ARCHIVE_DEST=/arch2/denver
REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE
```

The following table describes the archival processing shown in Figure 10–3:

	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/`.	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`	Is ignored; valid only when `chicago` 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 `/arch2/denver/`.
`LOG_ARCHIVE_DEST_3`	Directs transmission of redo data to the remote logical standby destination `denver`.	Is ignored; valid only when `denver` is running in the primary role.
`STANDBY_ARCHIVE_DEST`	Is ignored; valid only when `chicago` is running in the standby role.	Directs archiving of redo data received from the primary database directly to archived redo log files in `/arch2/denver/`.

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 10–3, 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 10–3, this is configured as the `LOG_ARCHIVE_DEST_2=LOCATION=/arch2/denver` destination.

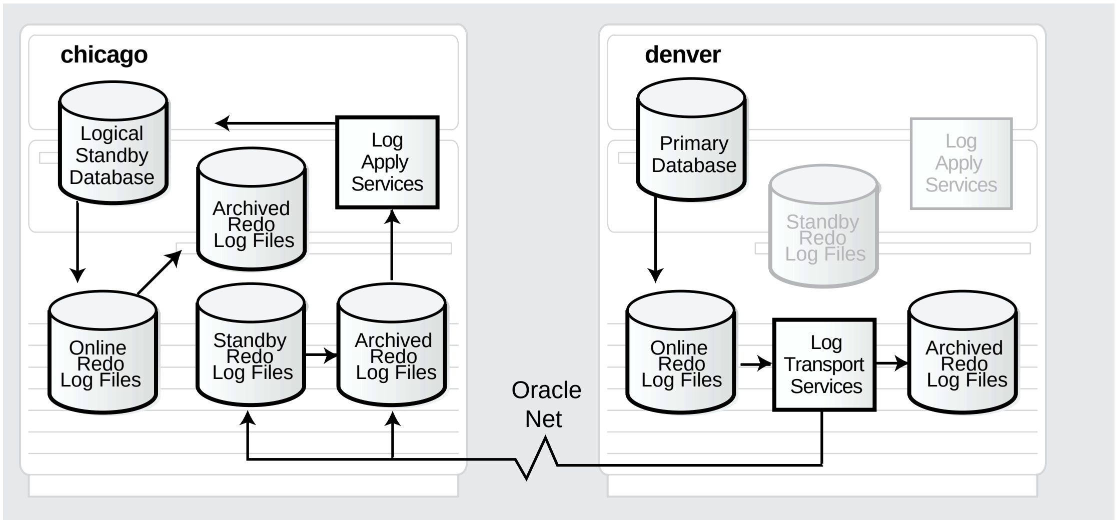
  In Figure 10–3, the `STANDBY_ARCHIVE_DEST` parameter is configured to the same location for these purposes:

  – 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 10–3 (and Figure 10–4) 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 10–4 shows the same configuration after a switchover.

Figure 10–4 Primary and Logical Standby Databases After a Role Transition



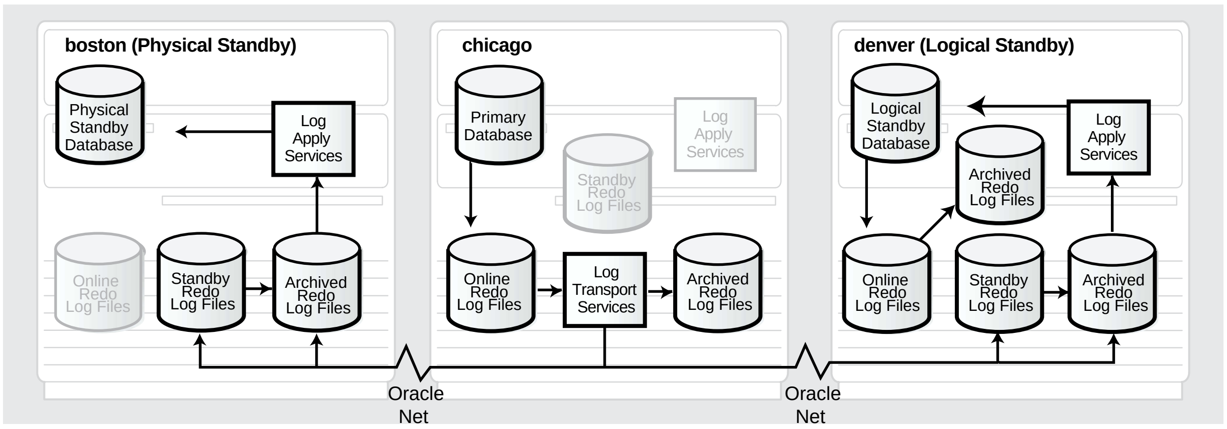
The following table describes the archival processing shown in Figure 10–4:

	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.

10.1.3 Configuring Both Physical and Logical Standby Databases

Figure 10–5 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 10–5 Configuring a Primary Database with Physical and Logical Standby Databases



```
DB_UNIQUE_NAME=boston
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_2=
 '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_2=ENABLE
LOG_ARCHIVE_DEST_STATE_3=ENABLE
STANDBY_ARCHIVE_DEST=
  /arch1/boston/
REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE
```

```
DB_UNIQUE_NAME=chicago
LOG_ARCHIVE_CONFIG=
 'DG_CONFIG=(chicago,boston,
denver)'
LOG_ARCHIVE_DEST_1=
 'LOCATION=/arch1/chicago/
  VALID_FOR=
   (ONLINE_LOGFILES,ALL_ROLES)
  DB_UNIQUE_NAME=chicago'
LOG_ARCHIVE_DEST_2=
 'SERVICE=denver
  VALID_FOR=
   (ONLINE_LOGFILES,PRIMARY_ROLE)
  DB_UNIQUE_NAME=denver'
LOG_ARCHIVE_DEST_3=
 'SERVICE=boston
  VALID_FOR=
   (ONLINE_LOGFILES,PRIMARY_ROLE)
  DB_UNIQUE_NAME=boston'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_STATE_2=ENABLE
LOG_ARCHIVE_DEST_STATE_3=ENABLE
STANDBY_ARCHIVE_DEST=
  /arch1/chicago/
REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE
```

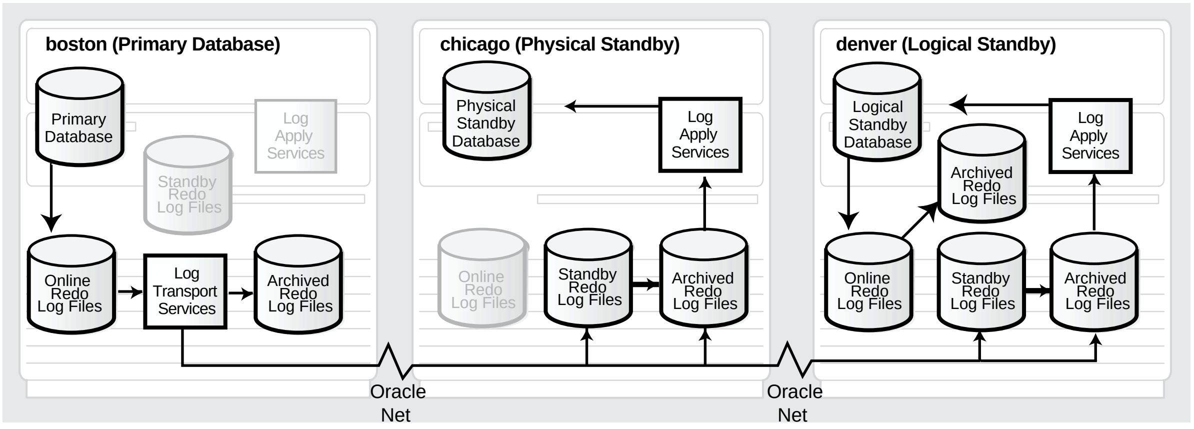
```
DB_UNIQUE_NAME=denver
LOG_ARCHIVE_CONFIG=
 'DG_CONFIG=(chicago,boston,
denver)'
LOG_ARCHIVE_DEST_1=
 'LOCATION=/arch1/denver/
  VALID_FOR=
   (ONLINE_LOGFILES,ALL_ROLES)
  DB_UNIQUE_NAME=denver'
LOG_ARCHIVE_DEST_2=
 'LOCATION=/arch2/denver/
  VALID_FOR=
   (STANDBY_LOGFILES,STANDBY_ROLE)
  DB_UNIQUE_NAME=denver'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_STATE_2=ENABLE
STANDBY_ARCHIVE_DEST=/arch2/denver
REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE
```

The following table describes the archival processing shown in Figure 10–5:

	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/`.
`LOG_ARCHIVE_DEST_3`	Directs transmission of redo data to the remote physical standby destination `boston`.	Is ignored; valid only when `boston` 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 `/arch1/boston/`.	Directs archiving of redo data received from the primary database directly to archived redo log files in `/arch2/denver/`.

Figure 10–6 shows the same configuration after a switchover changes the `chicago` database to the standby role and the `boston` database to the primary role.

Figure 10–6 Primary, Physical, and Logical Standby Databases After a Role Transition



The following table describes the archival processing shown in Figure 10–6:

	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 `/arch1/chicago/`.	Directs archival of redo data from the 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/denver/`.
`LOG_ARCHIVE_DEST_2`	Is ignored; valid only when `chicago` is running in the primary role.	Directs transmission of redo data to the remote logical standby destination `denver`.	Directs archival of redo data from the standby redo log files to the local archived redo log files in `/arch2/denver/`.
`LOG_ARCHIVE_DEST_3`	Is ignored; valid only when `chicago` is running in the primary role.	Directs transmission of redo data to the remote physical standby destination `chicago`.	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 `/arch1/chicago/`.	Is ignored; valid only for standby role.	Directs archival of redo data received from the primary database directly to archived redo log files in `/arch2/denver/`.

10.1.4 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 10–1.

Example 10–1 Finding VALID_FOR Information in the V$ARCHIVE_DEST View

```
SQL> SELECT DEST_10,VALID_TYPE,VALID_ROLE,VALID_NOW FROM V$ARCHIVE_DEST;
DEST_10  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 10–1, 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.

10.2 Choosing the Best Available Standby Database for a Role Transition

Every standby database is associated with only one primary database. A single primary database can, however, support multiple physical or logical standby databases. This scenario illustrates how to determine the information you need to choose the best available standby database for a failover or switchover.

If a configuration contains physical standby databases, Oracle recommends that you perform the role transition using the best available physical standby database if the environment uses both physical and logical standby databases. 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 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 at the standby location and by how much of these modifications were applied to 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 tradeoffs between additional risk of data loss and the time required to transition a standby database to the primary role.

10.2.1 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 the purpose of 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 10–2 provides information about the databases used in this scenario.

Table 10–2 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

Table 10–2 (Cont.) Identifiers for the Physical Standby Database Example

Identifier	HQ Database	SAT Database	NYC Database
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
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

See Appendix F for a sample ReadMe file.

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$ARCVHIVED_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$ARCVHIVED_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$ARCVHIVED_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$ARCVHIVED_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$ARCVHIVED_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.

10.2.2 Example: Best Logical Standby Database for a Failover Operation

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 the purpose of 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 10–3 provides information about each of these databases.

Table 10–3 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

Table 10–3 (Cont.) Identifiers for Logical Standby Database Example

Identifier	HQ Database	SAT Database	NYC Database
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 DBA_LOGSTDBY_PROGRESS view:

```
SQL> SELECT APPLIED_SCN, NEWEST_SCN FROM DBA_LOGSTDBY_PROGRESS;

APPLIED_SCN NEWEST_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. (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 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 DBA_LOGSTDBY_PROGRESS view:

```
SQL> SELECT APPLIED_SCN, NEWEST_SCN FROM DBA_LOGSTDBY_PROGRESS;
APPLIED_SCN NEWEST_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. (The line for log file in the example output is in bold typeface to assist you in reading the output. The actual output will not display bolding.) 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 `DBA_LOGSTDBY_PROGRESS` view to see if the value of the `APPLIED_SCN` column is equal to the value of the `NEWEST_SCN` column:

   ```
   SQL> SELECT APPLIED_SCN, NEWEST_SCN FROM DBA_LOGSTDBY_PROGRESS;

   APPLIED_SCN NEWEST_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 `DBA_LOGSTDBY_PROGRESS` view to see if the value of the `APPLIED_SCN` column is equal to the value of the `NEWEST_SCN` column:

   ```
   SQL> SELECT APPLIED_SCN, NEWEST_SCN FROM DBA_LOGSTDBY_PROGRESS;

   APPLIED_SCN NEWEST_SCN
   ----------- ----------
        144205     144205
   ```

   Because the SCN values match, you can 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.

10.3 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:

- Converting a Failed Primary Database into a Physical Standby Database
- Converting 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 Advanced User's Guide* for more information.

10.3.1 Converting 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 database (if necessary), mount the old primary database, 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;
```

The old primary database is now a new physical standby database and is referred to as such in the following steps.

Step 3 Mount the new physical standby database.

Perform the following steps on the new physical standby database:

1. Disable the Flashback Database feature. This deletes the flashback logs, which are obsolete after the standby control file is restored:

   ```
   SQL> ALTER DATABASE FLASHBACK OFF;
   ```

2. Create the standby control file and shut down the database:

   ```
   SQL> ALTER DATABASE CREATE STANDBY CONTROLFILE AS control_file_name;
   SQL> SHUTDOWN IMMEDIATE;
   ```

3. Issue operating system copy commands to replace the current control files with the new standby control file.
4. Mount the new physical standby database using the new standby control file.

   ```
   SQL> STARTUP MOUNT;
   ```

5. Ensure the listener is running:

   ```
   LSNRCTL STAT list_name;
   ```

6. Enable Flashback Database:

   ```
   SQL> ALTER DATABASE FLASHBACK ON;
   ```

Step 4 Restart log transport services 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 log 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:

```
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_n` initialization parameters so that log 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.

10.3.2 Converting 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 standby database, without re-creating the old primary database.

Step 1 Determine the SCN at which the old standby database became the primary database.

On the new primary database, determine the SCN at which the old standby database became the new primary database using the following query:

```
SQL> SELECT VALUE AS BECAME_PRIMARY_SCN FROM DBA_LOGSTDBY_PARAMETERS
  2> WHERE NAME = 'END_PRIMARY_SCN';
```

Step 2 Flash back the failed primary database.

To create a new logical standby database, shut down the database (if necessary), mount the old primary database, flash it back to the value for BECAME_PRIMARY_SCN that was 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 3 Open the database with the RESETLOGS option.

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

Step 4 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.

10.4 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.)

10.4.1 Flashing Back a Physical Standby Database

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 log apply services.

- To start Redo Apply on a physical standby database:

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

- To start SQL Apply on a logical standby database:

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

The standby database is now ready to receive and apply redo from the primary database.

10.4.2 Flashing Back a Logical Standby Database

The following steps describe how to avoid re-creating a logical standby database after you have issued the OPEN RESETLOGS statement on the primary database.

Step 1 Flash back the primary database.

On the primary database, execute the following SQL statements to flash back and then open the database with the RESETLOGS option:

```
SQL> FLASHBACK DATABASE TO TIMESTAMP <timestamp you want to flash back to>;
SQL> ALTER DATABASE OPEN RESETLOGS;
```

Step 2 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) FROM V$DATABASE;
```

Step 3 Stop SQL Apply.

On the logical standby database, stop SQL Apply:

```
SQL> ALTER DATABASE STOP LOGICAL STANDBY APPLY;
SQL> SELECT APPLIED_SCN FROM DBA_LOGSTDBY_PROGRESS;
```

If the APPLIED_SCN is less than the value of the <*resetlogs_change#-2*>, you do not need to flashback the standby database and can proceed to Step 6. This may happen if SQL Apply is running with a delay. Otherwise, flash back the standby database as described in Step 4.

Step 4 Flash back the logical standby database.

Issue the following SQL statements to flash back the logical standby database to the same time used to flash back the primary database:

```
SQL> SHUTDOWN;
SQL> STARTUP MOUNT EXCLUSIVE;
SQL> FLASHBACK DATABASE TO TIMESTAMP <time of primary database flashback>;
SQL> ALTER DATABASE OPEN READ ONLY;
SQL> SELECT APPLIED_SCN FROM DBA_LOGSTDBY_PROGRESS;
```

Step 5 Open the logical standby database with the RESETLOGS option.

Open the logical standby database with the RESETLOGS option:

```
SQL> SHUTDOWN;
SQL> STARTUP MOUNT EXCLUSIVE;
SQL> ALTER DATABASE OPEN RESETLOGS;
```

Step 6 Archive the current log on the primary database.

Perform a log switch:

```
SQL> ALTER SYSTEM ARCHIVE LOG CURRENT;
```

Step 7 Start SQL Apply.

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

10.5 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.

For example, suppose you run a batch job every night on the primary database. Unfortunately, you accidently 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.

10.5.1 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=stdby 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 transmitted to 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.

10.5.2 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 SKIP STANDBY LOGFILE;
   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.

10.5.3 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, 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;
SQL> STARTUP PFILE=Failover.ora;
```

10.6 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.

10.7 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.*

10.7.1 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 9.1.7.

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.

10.7.2 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 unjournaled 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/stdby/tbs_1.dbf                                      5186
/oracle/dbs/stdby/tbs_2.dbf                                         0
/oracle/dbs/stdby/tbs_3.dbf                                         0
/oracle/dbs/stdby/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 On the standby database, restart Redo Apply.

Issue the following SQL statement:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE DISCONNECT FROM
  2>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/stdby/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 3. See Section 5.8.4 for information about manually resolving an archive gap.

10.7.3 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.

10.8 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.

10.8.1 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:

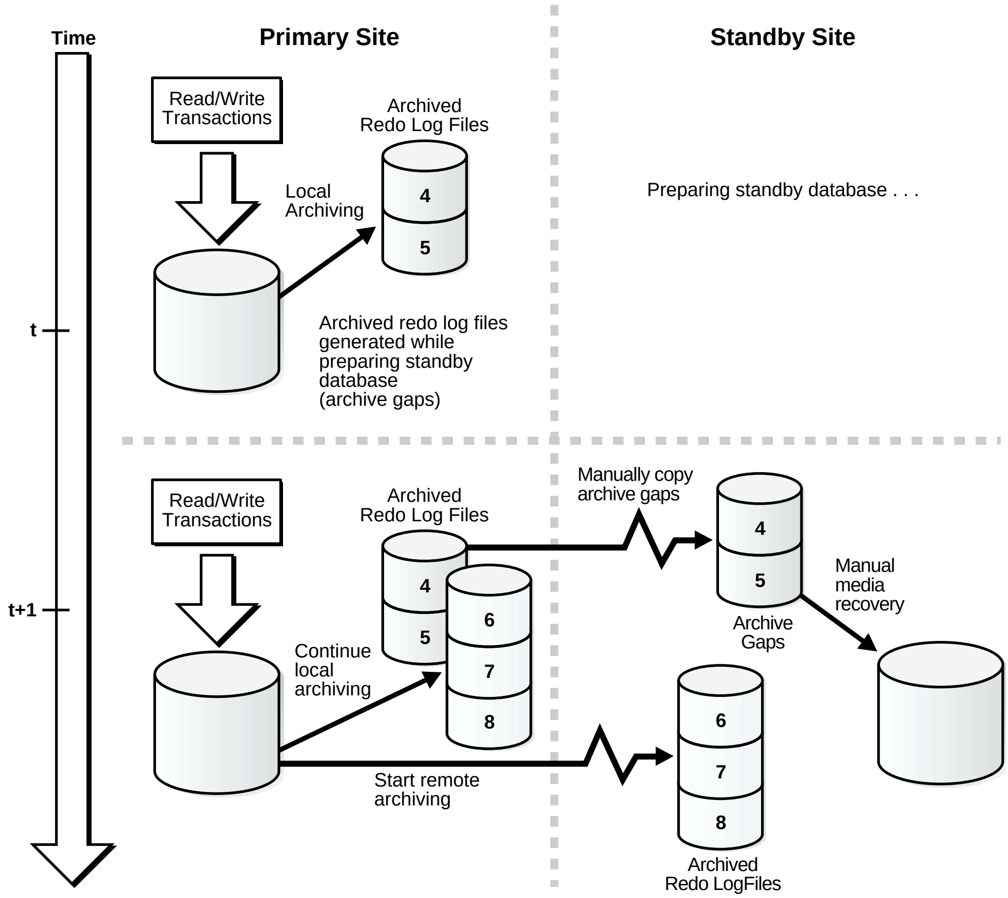
- Creation of the Standby Database
- Shutdown of the Standby Database When the Primary Database Is Open
- Network Failure Prevents Transmission of Archived Log Files

10.8.1.1 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 10–7.

Figure 10–7 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.

10.8.1.2 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.

10.8.1.3 Network Failure Prevents Transmission of Archived Log Files

If you maintain a Data Guard environment, and the network goes down, the primary database might continue to archive to disk but be unable to archive to the standby site. In this situation, archived redo log files accumulate as usual on the primary site, but the standby instance 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 10.6 for a related scenario

10.8.2 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
```

10.8.3 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_d.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 10.8.4 to apply the archive gap log files to the standby database. See also Section 6.3.4.3 and the V$ARCHIVED_LOG view in Chapter 14.

10.8.4 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;
```

10.9 Creating a Standby Database That Uses OMF or ASM

Chapters 3 and 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 D 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 Recovery Manager 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 4.2.4 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 10*g* documentation set.

This part contains the following chapters:

11 Initialization Parameters

This chapter describes the initialization parameters that affect databases in a Data Guard environment.

Table 11–1 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 set 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 11–1 Initialization Parameters for Instances in a Data Guard Configuration

Parameter	Primary Role?	Standby Role?	Notes and Recommendations
ARCHIVE_LAG_TARGET = *seconds*	Yes	No	Optional. Forces a log switch after the specified number of seconds elapses.
COMPATIBLE = *release_number*	Yes	Logical and physical	Data Guard requires a minimum value of 9.2.0.1.0. Set to a minimum of 10.0.0.0 to use Oracle Database 10*g* new features. Specify the same value on the primary and standby databases. If the values differ, log transport services may be unable to transmit redo data from the primary database to the standby databases. See Section 3.2.3 and Section 4.2.3.2 for examples.
CONTROL_FILE_RECORD_KEEP_TIME = *number_of_days*	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.
CONTROL_FILES = '*control_file_name*' , *control_file_name*', '...')	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 and Section 4.2.3.2 for examples.
DB_FILENAME_CONVERT = (*location_of_primary_database_datafile*' , '*location_of_standby_database_datafile_name*' , '...'	No	Logical and physical	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 3–2.
DB_FILES = *number_of_database_files_that_can_be_open_for_this_database*	Yes	Logical and physical	Optional. Specify the same number of database files on the primary and standby databases.

Table 11–1 (Cont.) Initialization Parameters for Instances in a Data Guard Configuration

Parameter	Primary Role?	Standby Role?	Notes and Recommendations
DB_NAME = *8-character_database_name*	Yes	Logical and physical	Required. For physical standby databases, specify the same DB_NAME that was specified for the primary database. For logical standby databases, use the DBNEWID utility to reset DB_NAME to specify a different name from the primary database, as described in Section 4.2.4.
DB_UNIQUE_NAME = *unique_service_provider_name_for_this_database*	Yes	Logical and physical	Required for remote destinations. Specify 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.
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 and Section 4.2.3.2 for examples.
LOG_ARCHIVE_CONFIG='DG_CONFIG=(*db_unique_name, db_unique_name, ...*)'	Yes	Logical and physical	Required. Specify the DG_CONFIG attribute to list the DB_UNIQUE_NAME for the primary database and each standby database in the Data Guard configuration. By default, 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.

Table 11–1 (Cont.) Initialization Parameters for Instances in a Data Guard Configuration

Parameter	Primary Role?	Standby Role?	Notes and Recommendations
LOG_ARCHIVE_DEST_*n* = {LOCATION=*path_name* | SERVICE=*service_name, attribute, attribute, ...* }	Yes	Logical and physical	Required. Define up to ten (where *n* = 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 12 for more information.
LOG_ARCHIVE_DEST_STATE_*n* = {ENABLE | DISABLE | ALTERNATE}	Yes	Logical and physical	Required. Specify a LOG_ARCHIVE_DEST_ STATE_*n* parameter to enable or disable log transport services to transmit redo data to the specified (or to an alternate) destination. Define a LOG_ARCHIVE_DEST_STATE_*n* parameter for every LOG_ARCHIVE_DEST_*n* parameter. See also Section 5.2.2 and Chapter 12.
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 (ARC*n*) transmit; either *after* (TRUE) the online redo log file was successfully archived to at least one local destination, or *at the same time* (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 10) of archiver processes you want Oracle software to invoke initially.
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 E.

Table 11–1 (Cont.) Initialization Parameters for Instances in a Data Guard Configuration

Parameter	Primary Role?	Standby Role?	Notes and Recommendations
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 and Section 4.2.3.2 for examples.
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. See also Section 9.4.
REMOTE_LOGIN_PASSWORDFILE= {EXCLUSIVE | SHARED}	Yes	Logical and physical	Required. Specify on the primary and all standby databases to ensure Data Guard transmits redo data only after the appropriate authentication checks using SYS credentials are successful. Include either the EXCLUSIVE or SHARED values to control how many databases can use the password file. See also Section 5.3.3.
SERVICE_NAMES	Yes	Yes	Specify a service name for this standby database that is unique from the primary database service name. If you do not explicitly specify unique service names and the primary and standby databases are located on the same system, the same default global name (consisting of the database name, DB_NAME, and domain name, DB_DOMAIN, parameters) will be in effect for both 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 11–1 (Cont.) Initialization Parameters for Instances in a Data Guard Configuration

Parameter	Primary Role?	Standby Role?	Notes and Recommendations
`STANDBY_ARCHIVE_DEST=` *filespec*	No	Logical and physical	Optional. Specify the location of archived redo log files on the standby database. The `STANDBY_ARCHIVE_DEST` initialization parameter overrides the directory location specified with the `LOG_ARCHIVE_DEST_`*n* 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	Logical and physical	Set the `STANDBY_FILE_MANAGEMENT` parameter to `AUTO` so that when data files 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 standby database. See Example 3–2 for more information and examples.
`USER_DUMP_DEST =` *directory_path_name_of_trace_file*	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 E.

12
LOG_ARCHIVE_DEST_n Parameter Attributes

This chapter provides syntax, values, and information on validity for the archival attributes of the `LOG_ARCHIVE_DEST_`*n* initialization parameter. The following list shows the attributes:

Each `LOG_ARCHIVE_DEST_`*n* destination you define must contain either a `LOCATION` or `SERVICE` attribute to specify a local disk directory or a remotely accessed database, respectively.

See Chapter 5 for information about defining `LOG_ARCHIVE_DEST_`*n* destinations to set up log transport services.

12.1 Changing Destination Attributes

You can set and 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 and ALTER SESSION statements. Table 12–1 lists the attributes that can be changed using an ALTER SYSTEM or ALTER SESSION statement.

Table 12–1 Changing Destination Attributes Using SQL

Attribute	**ALTER SYSTEM**	**ALTER SESSION**
[NO]AFFIRM	Yes	Yes
[NO]ALTERNATE=*destination*	Yes	Yes
ARCH	Yes	Yes
ASYNC[=*blocks*]	Yes	No
[NO]DELAY	Yes	Yes
[NO]DEPENDENCY=*destination*	Yes	No
LGWR	Yes	No
LOCATION=*local_disk_directory*	Yes	Yes
MANDATORY	Yes	Yes
[NO]MAX_FAILURE=*count*	Yes	No
OPTIONAL	Yes	Yes
[NO]NET_TIMEOUT[=*seconds*]	Yes	No
[NO]QUOTA_SIZE=*blocks*	Yes	No
[NO]QUOTA_USED=*blocks*	Yes	No
[NO]REGISTER	Yes	Yes
[NO]REOPEN[=*seconds*]	Yes	Yes
SERVICE=*net_service_name*	Yes	Yes
[NO]DB_UNIQUE_NAME	Yes	No
SYNC[=PARALLEL|NOPARALLEL]	Yes	Yes
[NO]TEMPLATE=*filename_template*	Yes	Yes
VALID_FOR	Yes	Yes
[NO]VERIFY	Yes	Yes

The modifications take effect after the next log switch on the primary database. For example, to defer log 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;
```

When updating attribute values in this way, you can incrementally change one or more additional attributes without having to re-specify the entire parameter value. For example, the following statements set a `REOPEN` attribute for the `LOG_ARCHIVE_DEST_2` destination, and set multiple attributes for the `LOG_ARCHIVE_DEST_1` destination incrementally on separate lines:

```
ALTER SYSTEM SET LOG_ARCHIVE_DEST_1='LOCATION=/arch1/chicago/';
ALTER SYSTEM SET LOG_ARCHIVE_DEST_2='REOPEN=60';
ALTER SYSTEM SET LOG_ARCHIVE_DEST_1='OPTIONAL';
```

Because specifying the `LOCATION` or `SERVICE` attribute causes the destination initialization parameter to be reset to its default values, note that the `SERVICE` or `LOCATION` attribute must be specified only on the first line. The statements are nonincremental because the `LOG_ARCHIVE_DEST_1` destination is reset each time:

```
ALTER SYSTEM SET LOG_ARCHIVE_DEST_1='LOCATION=/arch1/chicago/ REOPEN=60';
ALTER SYSTEM SET LOG_ARCHIVE_DEST_1='LOCATION=/arch1/chicago/';
```

To clear a previously entered destination specification, enter a null value:

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

12.2 Viewing Current Settings of Destination Initialization Parameters

Query the `V$ARCHIVE_DEST` view to see current settings of the `LOG_ARCHIVE_DEST_`*n* initialization parameter.

Note: Do not use the `V$PARAMETER` view to determine the value of the `LOG_ARCHIVE_DEST_`*n* parameter. The `V$PARAMETER` view shows only the last specified value for each parameter, which in the case of an incremental modification, is not representative of the actual `LOG_ARCHIVE_DEST_`*n* parameter value.

AFFIRM and NOAFFIRM

The AFFIRM and NOAFFIRM attributes control whether synchronous or asynchronous network I/O is used to write redo data to a remote standby redo log file or archived redo log file:

- AFFIRM—ensures all disk I/O to the archived redo log files or standby redo log files at the standby destination is performed synchronously and completes successfully before online redo log files on the primary database can be reused. The AFFIRM attribute is required to achieve no data loss.
- NOAFFIRM—indicates all disk I/O to archived redo log files and standby redo log files is to be performed asynchronously; online redo log files on the primary database can be reused before the disk I/O on the standby destination completes.

Note: 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.

If neither the AFFIRM nor the NOAFFIRM attribute is specified, the default is NOAFFIRM.

Category	AFFIRM	NOAFFIRM
Datatype of the attribute	Keyword	Keyword
Minimum attribute value	Not applicable	Not applicable
Maximum attribute value	Not applicable	Not applicable
Default attribute value	Not applicable	Not applicable
Requires attributes ...	Not applicable	Not applicable
Conflicts with attributes ...	NOAFFIRM	AFFIRM
Attribute class	ALTER SESSION and ALTER SYSTEM	ALTER SESSION and ALTER SYSTEM
Corresponding V$ARCHIVE_DEST column	AFFIRM	AFFIRM

Category	AFFIRM	NOAFFIRM
Related V$ARCHIVE_DEST column	ASYNC_BLOCKS	ASYNC_BLOCKS

AFFIRM

The AFFIRM attribute indicates all disk I/O to archived redo log files and standby redo log files is to be performed synchronously, even when the redo data is transmitted to a remote standby database. The AFFIRM attribute can be specified with either the LOCATION or SERVICE attributes for archival operations to local or remote destinations.

This attribute has the potential to 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.

Query the AFFIRM column of the V$ARCHIVE_DEST fixed view to see whether or not the AFFIRM attribute is being used for the associated destination.

Note: When the primary database is in the maximum protection or maximum availability mode, destinations using the log writer process are automatically placed in AFFIRM mode.

See also the SYNC and ASYNC attributes on page 12-43.

NOAFFIRM

The NOAFFIRM attribute indicates that all disk I/O to archived redo log files and standby redo log files is to be performed asynchronously; the LGWR process on the primary database does not wait until the disk I/O completes before continuing. The

NOAFFIRM attribute can be specified with either the LOCATION attribute for local destinations and with the SERVICE attribute for remote destinations.

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 and NOALTERNATE

These attributes control whether or not an alternate destination is used when the original archiving destination fails:

- ALTERNATE—defines an alternate archiving destination.
- NOALTERNATE—prevents archiving to an alternate destination.

If neither the ALTERNATE nor the NOALTERNATE attribute is specified, the default is NOALTERNATE. If the NOALTERNATE attribute is specified, or if no alternate destination is specified, the destination does not automatically change to another destination upon failure.

Category	**ALTERNATE=*LOG_ARCHIVE_DEST_n***	**NOALTERNATE**
Datatype of the attribute	String value	Keyword
Minimum attribute value	Not applicable	Not applicable
Maximum attribute value	Not applicable	Not applicable
Default attribute value	None	Not applicable
Requires attributes ...	Not applicable	Not applicable
Conflicts with attributes ...	NOALTERNATE	ALTERNATE
Attribute class	ALTER SYSTEM and ALTER SYSTEM	ALTER SESSION and ALTER SYSTEM
Corresponding V$ARCHIVE_DEST column	ALTERNATE	ALTERNATE
Related V$ARCHIVE_DEST column	STATUS	STATUS

ALTERNATE=*LOG_ARCHIVE_DEST_n*

The ALTERNATE attribute specifies another LOG_ARCHIVE_DEST_*n* destination that will be used if archival operations to the original destination fails. An alternate destination can reference either a local or remote archiving destination. For example, the following parameter specifies that if the LOG_ARCHIVE_DEST_1 destination fails, archival operations will automatically switch to the LOG_ARCHIVE_DEST_2 destination.

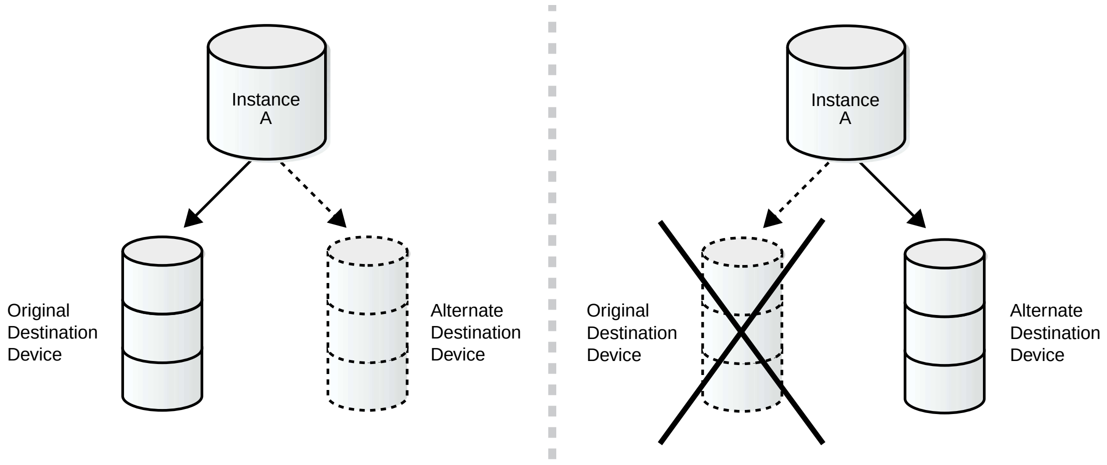
```
LOG_ARCHIVE_DEST_1='LOCATION=/disk1 MANDATORY ALTERNATE=LOG_ARCHIVE_DEST_2'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_2='LOCATION=/disk2 MANDATORY'
LOG_ARCHIVE_DEST_STATE_2=ALTERNATE
```

You can specify only one alternate destination for each `LOG_ARCHIVE_DEST_`*n* parameter. An alternate destination is used when the transmission of redo data from the primary site to the standby site fails. If it fails and the `REOPEN` attribute is specified with a value of zero (0), or `NOREOPEN` is specified, archival operations will attempt to transmit redo data to the alternate destination the next time redo data is archived.

A destination can also be in the `ALTERNATE` state; this state is specified using the `LOG_ARCHIVE_DEST_STATE_`*n* initialization parameter. The `ALTERNATE` state defers processing of the destination until such time as another destination failure automatically enables this destination, if the alternate destination attributes are valid. See Section 5.2.2 for information about the `LOG_ARCHIVE_DEST_STATE_`*n* parameter.

Figure 12–1 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 12–1 Archival Operation to an Alternate Destination Device


Instance A
Original Destination Device
Alternate Destination Device
Instance A
Original Destination Device
Alternate Destination Device

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

- The `NOREOPEN` attribute is specified.
- A value of zero (0) is specified for the `REOPEN` attribute.
- A nonzero `REOPEN` attribute and a nonzero `MAX_FAILURE` count were 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.

The following table shows the attribute precedences for standby destinations. In the left-most column, a 1 indicates highest precedence; 4 indicates lowest precedence.

Precedence	Attribute
1	`MAX_FAILURE`
2	`REOPEN`
3	`ALTERNATE`
4	`MANDATORY`

The use of a standby database as the target of an alternate destination should be carefully handled. Ideally, a standby alternate destination should only be used to specify a different network route to the same standby database system.

If no enabled destination references the alternate destination, the alternate destination is implied to be deferred, because there is no automatic method of enabling the alternate destination.

An alternate destination can be manually enabled at runtime. Conversely, an alternate destination can be manually deferred at runtime. See *Oracle Database Administrator's Guide* for more information about changing initialization parameter settings using SQL at runtime.

There is no general pool of alternate standby destinations. Ideally, for any enabled destination, the database administrator should choose an alternate destination that closely mirrors that of the referencing destination, although that is not required.

Each enabled destination can have its own alternate destination. Conversely, several enabled destinations can share the same alternate destination. This is known as an overlapping set of destinations. Enabling the alternate destination determines the set to which the destination belongs.

Increasing the number of enabled destinations decreases the number of available alternate archiving destinations.

Note: An alternate destination is enabled for 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.

Any destination can be designated as an alternate 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.

Destinations defined using the SQL ALTER SESSION statement do not activate an alternate destination defined at the system level. Conversely, system-defined destinations do not activate an alternate destination defined at the session level.

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.

NOALTERNATE

Use the NOALTERNATE attribute of the LOG_ARCHIVE_DEST_*n* parameter to prevent the original destination from automatically changing to an alternate destination when the original destination fails.

Examples

In the sample initialization parameter file in Example 12–1, 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 12–1 Automatically Failing Over to an Alternate Destination

```
LOG_ARCHIVE_DEST_1=
'LOCATION=/disk1 MANDATORY NOREOPEN ALTERNATE=LOG_ARCHIVE_DEST_2'
```

```
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_2='LOCATION=/disk2 MANDATORY'
LOG_ARCHIVE_DEST_STATE_2=ALTERNATE
```

The sample initialization parameter file in Example 12–2 shows how to define an alternate Oracle Net service name to the same standby database.

Example 12–2 Defining 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 NOREOPEN OPTIONAL ALTERNATE=LOG_ARCHIVE_DEST_3'
LOG_ARCHIVE_DEST_STATE_2=ENABLE
LOG_ARCHIVE_DEST_3='SERVICE=stby1_path2 NOREOPEN OPTIONAL'
LOG_ARCHIVE_DEST_STATE_3=ALTERNATE
```

ARCH and LGWR

The optional ARCH and LGWR attributes specify the process that will perform archival operations:

- ARCH—the archiver processes (ARC*n*) are responsible for transmitting redo data to archival destinations.
- LGWR—the log writer process (LGWR) is responsible for transmitting redo data to archival destinations.

By default, archiving is performed by ARC*n* processes; you must explicitly specify the LGWR attribute for log transport services to use the LGWR process. Although you cannot specify both LGWR and ARC*n* 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 ARC*n* process to the LGWR process), archival processing does not change until the next log switch occurs.

If neither the ARCH or LGWR attribute is specified, the default is ARCH.

Category	**ARCH**	**LGWR**
Datatype of the attribute	Keyword	Keyword
Minimum attribute value	Not applicable	Not applicable
Maximum attribute value	Not applicable	Not applicable
Default attribute value	Not applicable	Not applicable
Requires attributes ...	Not applicable	Not applicable
Conflicts with attributes ...	LGWR, ASYNC, NET_TIMEOUT	ARCH
Attribute class	ALTER SESSION and ALTER SYSTEM	ALTER SYSTEM
Corresponding V$ARCHIVE_DEST column	ARCHIVER	ARCHIVER
Related V$ARCHIVE_DEST columns	PROCESS, SCHEDULE	PROCESS, SCHEDULE

See the LOCATION and SERVICE attributes for information about controlling the transmission of redo data to local and remote standby destinations.

ARCH

The ARCH attribute indicates that archiver processes (ARC*n*) will transmit the current redo data to the associated destination when a redo log switch occurs on the primary database. As redo data is transmitted to the standby system, the RFS process writes the redo data to the archived redo log file and to the standby redo log file, if implemented. The ARCH attribute is the default setting.

When the ARCH attribute is specified for the destination, log transport services only perform synchronous network transmission. An error message is returned if you specify the ARCH and ASYNC attributes together.

LGWR

The LGWR attribute indicates that redo data is transmitted to the standby destination by the background LGWR process at the same time as it writes to the online redo log file on the primary database. As redo data is generated for the primary database, it is also propagated to the standby system where the RFS process writes the redo data to either a standby redo log file or an archived redo log file.

However, when you specify either the LGWR and ASYNC attributes or the LGWR and SYNC=PARALLEL attributes, the LGWR process uses a Network Server (LNS) process that transmits the redo data to the standby destination on behalf of the LGWR process. See Section 5.3.2 for more information.

When transmitting redo data to remote destinations, the LGWR process establishes a network connection to the destination instance. Because the redo data is transmitted concurrently, the redo data is not retransmitted to the corresponding destination during the archival operation. If a destination running in maximum availability or maximum performance mode fails, the destination automatically reverts to using the ARC*n* process until the problem is corrected.

Example

The following example shows the LGWR attribute with the LOG_ARCHIVE_DEST_*n* parameter. Section 5.3 provides more examples using these attributes.

```
LOG_ARCHIVE_DEST_3='SERVICE=denver LGWR'
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

DB_UNIQUE_NAME and NODB_UNIQUE_NAME

The DB_UNIQUE_NAME attribute specifies the database unique name for this destination. The DB_UNIQUE_NAME attribute must match the value that was defined originally for this database with the DB_UNIQUE_NAME initialization parameter.

There is no default value for this attribute.

Category	DB_UNIQUE_NAME=*name*	NODB_UNIQUE_NAME
Datatype of the attribute	String	String
Minimum attribute value	Not applicable	Not applicable
Maximum attribute value	Not applicable	Not applicable
Default attribute value	Not applicable	NODB_UNIQUE_NAME
Requires attributes ...	Not applicable	Not applicable
Conflicts with attributes ...	NODB_UNIQUE_NAME	DB_UNIQUE_NAME
Attribute class	ALTER SYSTEM	ALTER SYSTEM
Corresponding V$ARCHIVE_DEST column	DB_UNIQUE_NAME	DB_UNIQUE_NAME
Related V$ARCHIVE_DEST column	DB_UNIQUE_NAME	DB_UNIQUE_NAME

DB_UNIQUE_NAME=*name*

The DB_UNIQUE_NAME=*name* attribute must match the DB_UNIQUE_NAME initialization parameter of the database identified by the destination. If the LOG_ARCHIVE_CONFIG=DG_CONFIG parameter is not specified, the DB_UNIQUE_NAME attribute is optional. If the LOG_ARCHIVE_CONFIG=DG_CONFIG parameter is specified, the DB_UNIQUE_NAME attribute:

- Is required for remote destinations (specified with the SERVICE attribute) and must match one of the DB_UNIQUE_NAME values in the DG_CONFIG list. Furthermore, log transport services validates that the DB_UNIQUE_NAME of the database at the specified destination matches the DB_UNIQUE_NAME attribute or the connection to that destination is refused.

- Is optional for local destinations (specified with the LOCATION attribute). However, when you specify a local destination, the name you specify with the DB_UNIQUE_NAME attribute must match the name specified for the database's DB_UNIQUE_NAME initialization parameter.

NODB_UNIQUE_NAME

If you specify the NODB_UNIQUE_NAME attribute and the LOG_ARCHIVE_CONFIG parameter is not defined, this will reset the database unique name for the destination. That is, the NODB_UNIQUE_NAME attribute clears any value that you previously specified with the DB_UNIQUE_NAME attribute. The NODB_UNIQUE_NAME attribute is not valid if the LOG_ARCHIVE_CONFIG parameter is defined.

Example

The following example is a portion of a text initialization parameter file showing how to specify the DB_UNIQUE_NAME attribute on the LOG_ARCHIVE_DEST_*n* parameter. The definitions for the DB_UNIQUE_NAME and the LOG_ARCHIVE_CONFIG initialization parameters are provided to add clarity.

In the example, the DB_UNIQUE_NAME for this database is 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_STATE_1=ENABLE
LOG_ARCHIVE_DEST_2='SERVICE=Sales_DR VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) DB_UNIQUE_
NAME=chicago'
LOG_ARCHIVE_DEST_STATE_2=ENABLE
.
.
.
```

DELAY and NODELAY

When log apply services are enabled on a physical standby database, redo data is written to archived redo log files or standby redo log files and then applied (except when real-time apply is enabled which allows Data Guard to recover redo data from the current standby redo log file as it is being filled up). However, a DELAY attribute, which specifies a time lag between archiving redo data on the standby site and applying the archived redo log file to the standby database, may be used to protect the standby database from corrupted or erroneous primary data.

Note: You can set this attribute only for physical standby databases. To delay the application of archived redo log files on a logical standby databases, use the DBMS_LOGSTDBY.APPLY_SET procedure as described in *PL/SQL Packages and Types Reference*.

If neither the DELAY nor the NODELAY attribute is specified, the default is NODELAY.

Category	DELAY[=*minutes*]	NODELAY
Datatype of the attribute	Numeric	Keyword
Minimum attribute value	0 minutes	Not applicable
Maximum attribute value	Unlimited	Not applicable
Default attribute value	30 minutes	Not applicable
Requires attributes ...	SERVICE	Not applicable
Conflicts with attributes ...	LOCATION, NODELAY	DELAY
Attribute class	ALTER SESSION and ALTER SYSTEM	ALTER SESSION and ALTER SYSTEM
Corresponding V$ARCHIVE_DEST column	DELAY_MINS	DELAY_MINS
Related V$ARCHIVE_DEST column	DESTINATION	DESTINATION

DELAY[=*minutes*]

Use the DELAY attribute of the LOG_ARCHIVE_DEST_*n* initialization parameter to specify a time lag for the application of archived redo log files to the physical standby database. The DELAY attribute does not affect the transmittal of redo data to the physical standby destination. If you have real-time apply enabled, any delay that you set will be ignored.

> **Note:** Changes to the DELAY attribute take effect the next time redo data is archived. In-progress archiving is not affected.

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.

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. For example, assume primary database A supports standby databases B, C, and D. Standby database B is set up as the disaster recovery database and therefore has no time lag. Standby database C is set up to protect against logical or physical corruption, and is maintained with a 2-hour delay. Standby database D is maintained with a 4-hour delay and protects against further corruption.

You can override the specified delay interval at the standby site. To immediately apply an archived redo log file to the standby database before the time interval expires, use the NODELAY keyword of the RECOVER MANAGED STANDBY DATABASE clause. For example:

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

NODELAY

When you specify the NODELAY attribute and Redo Apply is enabled on the physical standby database, archived redo log files are applied when a log switch occurs on the primary database.

See *Oracle Database SQL Reference* for information about the DELAY attribute on the ALTER DATABASE RECOVER MANAGED STANDBY DATABASE statement.

Examples

The following example shows the DELAY attribute with the LOG_ARCHIVE_DEST_*n* parameter.

```
LOG_ARCHIVE_DEST_3='SERVICE=stby1 DELAY=240'
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

DEPENDENCY and NODEPENDENCY

The DEPENDENCY attribute transmits redo data to a destination that then shares its archived redo log files among multiple standby databases.

- DEPENDENCY—defines one archival destination to receive redo data on behalf of several destinations.
- NODEPENDENCY—specifies there is no dependency on the success or failure of an archival operation to another destination.

The transmission of redo data to the remote destination makes the *child* destinations dependent upon the success or failure of an archival operation to the *parent* destination.

If neither the DEPENDENCY nor the NODEPENDENCY attribute is specified, the default is NODEPENDENCY.

Category	**DEPENDENCY=*destination***	**NODEPENDENCY**
Datatype of the attribute	String value	Keyword
Minimum attribute value	Not applicable	Not applicable
Maximum attribute value	Not applicable	Not applicable
Default attribute value	Not applicable	Not applicable
Requires attributes ...	SERVICE, REGISTER	Not applicable
Conflicts with attributes ...	NODEPENDENCY, LOCATION, NOREGISTER, QUOTA_SIZE, QUOTA_USED	DEPENDENCY
Attribute class	ALTER SYSTEM	ALTER SYSTEM
Corresponding V$ARCHIVE_DEST column	DEPENDENCY	DEPENDENCY
Related V$ARCHIVE_DEST columns	Not applicable	Not applicable

DEPENDENCY=*destination*

Specify the DEPENDENCY attribute to define a local destination, a physical standby database, or a logical standby database. Specifying a destination dependency can be useful in the following configurations:

- The standby database and the primary database are on the same node. Therefore, the archived redo log files are implicitly accessible to the standby database.
- Clustered file systems provide remote standby databases with access to the primary database archived redo log files.
- Operating system-specific network file systems provide remote standby databases with access to the primary database archived redo log files.
- Mirrored disk technology provides transparent networking support across geographically remote distances.
- Multiple standby databases are on the same remote system, sharing access to common archived redo log files.

In these situations, although a physical archival operation does not occur for the dependent destination, the standby database needs to know the location of the archived redo log files. This allows the standby database to access the archived redo log files when they become available.

Consider the case of a two-node cluster where a primary node shares access to the destination with the standby node through a mirrored disk device. This configuration, where you maintain a local standby database, is useful for off-loading ad hoc queries and reporting functions.

The primary database archives an online redo log file locally and, upon successful completion, the archived redo log file is immediately available to the standby database for Redo Apply by a physical standby database. This does not require a physical remote archival operation for the standby destination. In this case, two destinations are used: one for local archiving and another for archiving at the standby site. The standby destination is not valid unless the primary destination succeeds. Therefore, the standby destination has a dependency upon the success or failure of the local destination.

Restrictions

The `DEPENDENCY` attribute has the following restrictions:

- Only standby destinations can have a dependency.
- The parent destination can be either a local or standby destination.
- The `DEPENDENCY` attribute cannot be modified at the session level.
- The `REGISTER` attribute is required.
- The `SERVICE` attribute is required.

When one or more destinations are dependent upon the same parent destination, all attributes of the dependent destinations still apply to that destination. It appears as if the archival operation was performed for each destination, when only one archival operation actually occurred.

Consider, for example, that two standby databases are dependent upon the same parent 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.

Note: Dependent destinations do not participate in a Data Guard no-data-loss environment.

NODEPENDENCY

Specifies there is no dependency on the success or failure of an archival operation to another destination.

Examples

One reason to use the DEPENDENCY attribute is if the standby database is on the same site as the primary database. Using this configuration, you only need to archive the redo data once and, because the standby database resides on the local system, it can access the same archived redo log files. The following is an example of the LOG_ARCHIVE_DEST_*n* parameters in this scenario:

```
# Set up the mandatory local destination:
#
LOG_ARCHIVE_DEST_1='LOCATION=/oracle/dbs/ MANDATORY'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
#
# Set up the dependent standby database that resides on the local system:
#
LOG_ARCHIVE_DEST_2='SERVICE=dest2 DEPENDENCY=LOG_ARCHIVE_DEST_1 OPTIONAL'
LOG_ARCHIVE_DEST_STATE_2=ENABLE
```

Another 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. The following is an example of this scenario:

```
# 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 log transport services can transmit redo data. For each Data Guard configuration, 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.

Either the LOCATION or the SERVICE attribute must be specified. There is no default. The LOCATION attribute conflicts with QUOTA_SIZE and QUOTA_USED only when the USE_DB_RECOVERY_FILE_DEST value is specified on the LOCATION attribute.

Category	**LOCATION=*local_disk_directory* or USE_DB_RECOVERY_FILE_DEST**	**SERVICE=*net_service_name***
Datatype	String value	String value
Minimum attribute value	Not applicable	Not applicable
Maximum attribute value	Not applicable	Not applicable
Default attribute value	Not applicable	Not applicable
Requires attributes. . .	Not applicable	Not applicable
Conflicts with attributes ...	SERVICE, DELAY, DEPENDENCY, NOREGISTER, ASYNC, TEMPLATE, NET_TIMEOUT, QUOTA_SIZE, QUOTA_USED	LOCATION, QUOTA_USED, QUOTA_SIZE
Attribute class	ALTER SESSION and ALTER SYSTEM	ALTER SESSION and ALTER SYSTEM
Corresponding V$ARCHIVE_DEST column	DESTINATION	DESTINATION

Category	LOCATION=*local_disk_directory* or USE_DB_RECOVERY_FILE_DEST	SERVICE=*net_service_name*
Related V$ARCHIVE_DEST column	TARGET	TARGET

Note: If you are specifying multiple attributes, specify the LOCATION or SERVICE attribute first in the list of attributes.

To verify the current settings for LOCATION and SERVICE, query the V$ARCHIVE_DEST fixed view:

- The TARGET column of the V$ARCHIVE_DEST fixed view identifies if the destination is local or remote to the primary database.
- The DESTINATION column of the V$ARCHIVE_DEST fixed view 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.

LOCATION=*local_disk_directory*

When you specify a LOCATION attribute, you can specify one of the following:

- LOCATION=*local_disk_directory*

  This specifies a valid path name for a disk directory on the system that hosts the database. Each destination that specifies the LOCATION attribute must identify a unique directory path name. This is the local destination for archived redo log files.

  Local destinations indicate that the archived redo log files are to reside within the file system that is accessible to the local database. Local archived redo log files remain physically within the primary database namespace. The destination parameter value specifies the local file system directory path where the log files are copied.

- LOCATION=USE_DB_RECOVERY_FILE_DEST

  To configure a flash recovery area, you specify the directory, file system, 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, Data Guard implicitly uses the LOG_ARCHIVE_DEST_

10 destination as the default disk location for the flash recovery area and for storing the archived redo log files. See Section 5.2.3 for more information about flash recovery areas.

Note: When you issue the ALTER DATABASE ARCHIVELOG statement to enable the archival process, it automatically derives the local archiving destination from the LOCATION attribute of the LOG_ARCHIVE_DEST_*n* parameter.

SERVICE=*network_service_name*

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.

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.

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 *Oracle Net Services Administrator's Guide* for details about setting up Oracle Net service names.

Examples

The following example shows the LOCATION attribute with the LOG_ARCHIVE_DEST_*n* parameter:

```
LOG_ARCHIVE_DEST_2='LOCATION=/disk1/oracle/oradata/payroll/arch/'
LOG_ARCHIVE_DEST_STATE_2=ENABLE
```

The following example shows the SERVICE attribute with the LOG_ARCHIVE_DEST_*n* parameter:

```
LOG_ARCHIVE_DEST_3='SERVICE=stby1'
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

MANDATORY and OPTIONAL

You can specify a policy for reusing online redo log files using the OPTIONAL or MANDATORY attributes. 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.

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 to be optional.

Category	MANDATORY	OPTIONAL
Datatype of the attribute	Keyword	Keyword
Minimum attribute value	Not applicable	Not applicable
Maximum attribute value	Not applicable	Not applicable
Default attribute value	Not applicable	Not applicable
Requires attributes ...	Not applicable	Not applicable
Conflicts with attributes ...	OPTIONAL	MANDATORY
Attribute class	ALTER SESSION and ALTER SYSTEM	ALTER SESSION and ALTER SYSTEM
Corresponding V$ARCHIVE_DEST column	BINDING	BINDING
Related V$ARCHIVE_DEST columns	Not applicable	Not applicable

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. 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
```

When determining how to set your parameters, note that:

- This attribute does not affect the data protection mode for the destination.
- 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 destinations, nor greater than the number of mandatory destinations plus the number of optional local destinations.
- If you defer a mandatory destination, and the online redo log file is overwritten without transferring the redo data to the standby site, then you must transfer the redo log file to the standby site manually.

The BINDING column of the V$ARCHIVE_DEST fixed view specifies how failure affects the archival operation.

MANDATORY

Specifies that the transmission of redo data to the destination must succeed before the local online redo log file can be made available for reuse.

OPTIONAL

Specifies that successful transmission of redo data to the destination is not required before the online redo log file can be made available for reuse. 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 marked for reuse.

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=denver MANDATORY'
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

MAX_FAILURE and NOMAX_FAILURE

These attributes control the number of times log transport services will attempt to reestablish communication to a failed destination.

- MAX_FAILURE—the maximum number of reopen attempts before the primary database permanently gives up on the standby database.
- NOMAX_FAILURE—allows an unlimited number of consecutive attempts to transport archive redo log files to the failed destination.

If neither the MAX_FAILURE nor the NOMAX_FAILURE attribute is specified, the default is NOMAX_FAILURE.

Category	**MAX_FAILURE=*count***	**NOMAX_FAILURE**
Datatype of the attribute	Numeric	Keyword
Minimum attribute value	0	Not applicable
Maximum attribute value	None	Not applicable
Default attribute value	None	Not applicable
Requires attributes ...	REOPEN	Not applicable
Conflicts with attributes ...	NOMAX_FAILURE	MAX_FAILURE
Dynamically changed by SQL statement . . .	ALTER SYSTEM	ALTER SYSTEM
Corresponding V$ARCHIVE_DEST column	MAX_FAILURE	MAX_FAILURE
Related V$ARCHIVE_DEST columns	FAILURE_COUNT, REOPEN_SECS	Not applicable

MAX_FAILURE=*count*

The MAX_FAILURE attribute specifies the maximum number of consecutive times log transport services attempt to transmit redo data to a failed destination. It limits the number of times log transport services attempt to reestablish communication and resume sending redo data to a failed destination. When you specify the MAX_FAILURE attribute, you must also set the REOPEN attribute to limit the number of consecutive attempts that will be made to reestablish communication with a failed destination. Once the specified number of consecutive attempts is exceeded, the destination is treated as if the NOREOPEN attribute was specified.

Using this attribute, you can provide 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 to specify how often archiving is retried to the particular destination.

To Limit the Number of Retry Attempts If you set both the MAX_FAILURE and REOPEN attributes to nonzero values, log transport services limits the number of archival attempts to the number of times specified by the MAX_FAILURE attribute. Each destination contains an internal failure counter that tracks the number of consecutive archival failures that have occurred. 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.

If an archival operation fails for any reason, the failure count is incremented until:

- The failure no longer occurs and archiving resumes.
- The failure count is greater than or equal to the value set for the MAX_FAILURE attribute.

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).

- You issue the ALTER SYSTEM SET statement to dynamically change the MAX_FAILURE attribute (or any other destination attribute). 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.

Note: Runtime modifications made to the destination, such as changing the QUOTA_USED attribute, do not affect the failure count.

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 the value zero (0), which causes log transport services to transport redo data to an alternate destination (defined with the ALTERNATE attribute) on the next archival operation.

To Attempt Archival Operations Indefinitely Log 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 or the NOMAX_FAILURE attribute), 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 in the event of a repeated failure.

NOMAX_FAILURE

Specify the NOMAX_FAILURE attribute to allow an unlimited number of archival attempts to the failed destination.

The NOMAX_FAILURE attribute is equivalent to specifying MAX_FAILURE=0.

Examples

The following example allows log 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 NOREOPEN attribute was specified.

```
LOG_ARCHIVE_DEST_1='LOCATION=/arc_dest REOPEN=5 MAX_FAILURE=3'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
```

NET_TIMEOUT and NONET_TIMEOUT

The NET_TIMEOUT and NONET_TIMEOUT attributes determine how long the log writer process waits before terminating the network connection:

- 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.
- NONET_TIMEOUT — reverses or undoes the timeout value that was previously specified with the NET_TIMEOUT attribute.

If you do not specify the NET_TIMEOUT attribute (or if you specify the NONET_TIMEOUT attribute), 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.

If neither the NET_TIMEOUT nor the NONET_TIMEOUT attribute is specified, the default is NONET_TIMEOUT.

Category	**NET_TIMEOUT=*seconds***	**NONET_TIMEOUT**
Datatype of the attribute	Numeric	Not applicable
Minimum attribute value	1[1]	Not applicable
Maximum attribute value	1200	Not applicable
Default attribute value	180 seconds	Not applicable
Requires attributes ...	LGWR with SYNC=PARALLEL or LGWR with ASYNC > 0	Not applicable
Conflicts with attributes ...	ARCH, LOCATION, NONET_TIMEOUT, LGWR with SYNC=NOPARALLEL, LGWR with ASYNC=0	NET_TIMEOUT
Attribute class	ALTER SYSTEM	ALTER SYSTEM
Corresponding V$ARCHIVE_DEST column	NET_TIMEOUT	NET_TIMEOUT
Related V$ARCHIVE_DEST column	Not applicable	Not applicable

[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.

NET_TIMEOUT=*seconds*

The NET_TIMEOUT attribute is used only when the log writer process transmits redo data using a network server (LNS*n*) process and when either the ASYNC or the SYNC=PARALLEL attribute is specified.

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 value of the Oracle Net EXPIRE_TIME parameter or keep alive intervals on the standby databases.

  Even though the network connection might be terminated on the primary database, the network connection remains active on the standby database until the corresponding TCP/IP network timers expire. For this reason, you need to coordinate the setting for the NET_TIMEOUT attribute on the primary database with the setting of the Oracle Net EXPIRE_TIME parameter on each standby database.

  Note: In general, you should set the Oracle Net EXPIRE_TIME parameter on the standby systems to expire before the timeout period specified by the NET_TIMEOUT attribute on the primary system. The EXPIRE_TIME parameter is expressed in minutes.

- 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 for approximately 15 seconds.
  - In maximum performance mode, the log writer process tries to reconnect for approximately 15 seconds.

For example, a primary database operating in the maximum availability protection mode with a NET_TIMEOUT attribute value set to 60 seconds and an EXPIRE_TIME of 1 minute 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 Net Services Administrator's Guide.*

NONET_TIMEOUT

The NONET_TIMEOUT attribute implies the log writer process waits for the default network timeout interval established for the system. The default network timeout interval differs from system to system.

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=PARALLEL'
LOG_ARCHIVE_DEST_STATE_2=ENABLE
```

QUOTA_SIZE and NOQUOTA_SIZE

The QUOTA_SIZE and the NOQUOTA_SIZE attributes of the LOG_ARCHIVE_DEST_*n* parameter indicate the maximum number of 512-byte blocks of physical storage on a disk device that can be used by a local destination.

If neither the QUOTA_SIZE nor the NOQUOTA_SIZE attribute is specified, the default is NOQUOTA_SIZE. The LOCATION attribute conflicts with QUOTA_SIZE and QUOTA_USED only when the USE_DB_RECOVERY_FILE_DEST value is specified on the LOCATION attribute.

Category	QUOTA_SIZE=*blocks*	NOQUOTA_SIZE
Datatype of the attribute	Numeric	Keyword
Minimum attribute value	0 blocks	Not applicable
Maximum attribute value	Unlimited blocks	Not applicable
Default attribute value	Not applicable	Not applicable
Requires attributes ...	LOCATION	Not applicable
Conflicts with attributes ...	NOQUOTA_SIZE, DEPENDENCY, SERVICE, LOCATION	QUOTA_SIZE
Attribute class	ALTER SYSTEM	ALTER SYSTEM
Corresponding V$ARCHIVE_DEST column	QUOTA_SIZE	QUOTA_SIZE
Related V$ARCHIVE_DEST column	QUOTA_USED	QUOTA_USED

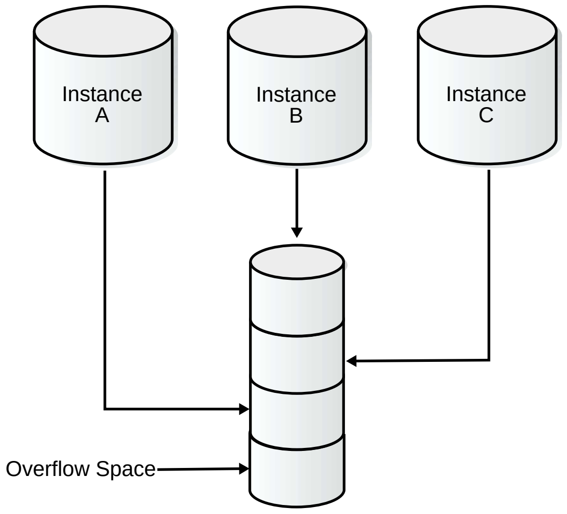
QUOTA_SIZE=*blocks*

The QUOTA_SIZE attribute indicates the maximum number of 512-byte blocks of physical storage on a disk device that might be used by a local destination. The value is specified in 512-byte blocks even if the physical device uses a different block size. The optional suffix values K, M, and G represent thousand, million, and billion, respectively (the value 1K means 1,000 512-byte blocks).

A local archiving destination can be designated as being able to occupy all or some portion of the physical disk. For example, in a Real Application Clusters environment, a physical archived redo log file's disk device might be shared by two or more separate nodes (through a clustered file system, such as is available with Sun Clusters). As there is no cross-instance initialization parameter knowledge, none of the Real Application Clusters nodes is aware that the archived redo log file's physical disk device is shared with other instances. This can lead to significant 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.

For example, consider an 8-gigabyte (GB) disk device `/dev/arc_dest` that is further subdivided into node-specific directories: `node_a`, `node_b`, and `node_c`. The DBA could designate that each of these instances is allowed to use a maximum of 2 GB, which would allow an additional 2 GB for other purposes. This scenario is shown in Figure 12–2.

Figure 12–2 Specifying Disk Quota for a Destination



No instance uses more than its allotted quota.

The quota is common to all users of the destination, including foreground archival operations, archiver processes, and even the log writer process.

Oracle highly recommends that the `ALTERNATE` attribute be used in conjunction with the `QUOTA_SIZE` attribute. However, this is not required.

See also the ALTERNATE and NOALTERNATE attributes on page 12-7.

NOQUOTA_SIZE

Use of the `NOQUOTA_SIZE` attribute, or the `QUOTA_SIZE` attribute with a value of zero (0), indicates that there is unlimited use of the disk device by this destination; this is the default behavior.

Examples

The following example shows the `QUOTA_SIZE` attribute with the `LOG_ARCHIVE_DEST_`*n* parameter.

```
LOG_ARCHIVE_DEST_4='QUOTA_SIZE=100K'
```

QUOTA_USED and NOQUOTA_USED

The QUOTA_USED and the NOQUOTA_USED attributes of the LOG_ARCHIVE_DEST_*n* parameter identify the number of 512-byte blocks of data that were archived on a specified destination.

If neither the QUOTA_USED nor the NOQUOTA_USED attribute is specified, the default is NOQUOTA_USED. The QUOTA_USED attribute has a default value of zero (0) for remote archival destinations. The LOCATION attribute conflicts with QUOTA_SIZE and QUOTA_USED only when the USE_DB_RECOVERY_FILE_DEST value is specified on the LOCATION attribute.

Category	**QUOTA_USED=*blocks***	**NOQUOTA_USED**
Datatype of the attribute	Numeric	Keyword
Minimum attribute value	0 blocks	Not applicable
Maximum attribute value	Unlimited blocks	Not applicable
Default attribute value	Not applicable	Not applicable
Requires attributes ...	LOCATION	Not applicable
Conflicts with attributes ...	NOQUOTA_USED, DEPENDENCY, SERVICE, LOCATION	QUOTA_USED
Attribute class	ALTER SYSTEM	ALTER SYSTEM
Corresponding V$ARCHIVE_DEST column	QUOTA_USED	QUOTA_USED
Related V$ARCHIVE_DEST column	QUOTA_SIZE	QUOTA_SIZE

QUOTA_USED=*blocks*

The QUOTA_USED attribute identifies the number of 512-byte blocks of data that were archived on the specified local destination. The value is specified in 512-byte blocks even if the physical device uses a different block size. The optional suffix values K, M, and G represent thousand, million, and billion, respectively (the value 1K means 1,000 512-byte blocks).

This attribute cannot be modified at the session level.

If you specify a QUOTA_SIZE attribute value greater than zero (0) for a destination, but do not specify a QUOTA_USED attribute value in the database initialization parameter file, the QUOTA_USED attribute value is automatically determined when the database is initially mounted. The QUOTA_USED attribute value defaults to the actual number of blocks residing on the local archiving destination device. If the calculated QUOTA_USED attribute value exceeds the QUOTA_SIZE attribute value, the QUOTA_SIZE attribute value is automatically adjusted to reflect the actual storage used. This automatic calculation of the QUOTA_USED value applies only to local archiving destinations.

Note: The runtime value of the QUOTA_USED attribute changes automatically as archival operations are started. The QUOTA_USED attribute value is automatically preallocated against the destination quota size. You do not need to change the value of this attribute.

If, at runtime, you dynamically modify the QUOTA_SIZE attribute value, but not the QUOTA_USED attribute value, the QUOTA_USED attribute value is not automatically recalculated.

For local destinations, the QUOTA_USED attribute value is incremented at the start of an archival operation. If the resulting value is greater than the QUOTA_SIZE attribute value, the destination status is changed to FULL, and the destination is rejected before the archival operation begins.

The QUOTA_SIZE and QUOTA_USED attributes are very important because they can be used together to detect a lack of disk space before the archival operation begins. Consider the case where the QUOTA_SIZE attribute value is 100K and the QUOTA_USED attribute value is 100K also. The destination status is VALID at this point. However, an attempt to archive 1 block results in the QUOTA_USED attribute value being changed to 101K, which exceeds the QUOTA_SIZE attribute value. Therefore, the destination status is changed to FULL, and the destination is rejected before the archival operation begins.

NOQUOTA_USED

Specifies that an unlimited number of blocks of data can be archived on a specified destination.

Examples

Data Guard automatically sets this value. You do not need to change the value of the QUOTA_USED and the NOQUOTA_USED attributes.

REGISTER and NOREGISTER

The REGISTER and the NOREGISTER attributes of the LOG_ARCHIVE_DEST_*n* parameter indicate if the location of the archived redo log file is to be recorded at the destination site.

If neither the REGISTER nor the NOREGISTER attribute is specified, the default is REGISTER.

Category	REGISTER	NOREGISTER
Datatype of the attribute	Keyword	Keyword
Minimum attribute value	Not applicable	Not applicable
Maximum attribute value	Not applicable	Not applicable
Default attribute value	Not applicable	Not applicable
Requires attributes ...	Not applicable	SERVICE
Conflicts with attributes ...	NOREGISTER, NOALTERNATE	REGISTER, LOCATION
Attribute class	ALTER SESSION and ALTER SYSTEM	ALTER SESSION and ALTER SYSTEM
Corresponding V$ARCHIVE_DEST column	DESTINATION	DESTINATION
Related $ARCHIVE_DEST column	TARGET	TARGET

REGISTER

The REGISTER attribute indicates that the location of the archived redo log file is to be recorded at the corresponding destination.

For a physical standby destination, the name of the archived redo log file is recorded in the destination database control file, which is then used by Redo Apply.

For a logical standby database, the name of the archived redo log file is recorded in the tablespace maintained by the logical standby database control file, which is then used by SQL Apply.

The REGISTER attribute implies that the destination is a Data Guard standby database.

By default, the location of the archived redo log file, at a remote destination, is derived from the destination's STANDBY_ARCHIVE_DEST and LOG_ARCHIVE_FORMAT initialization parameters.

Note: You can also set the REGISTER attribute by executing the SQL ALTER DATABASE REGISTER LOGFILE *filespec* statement on each standby database. See Section 7.2.2.1 for an example of this SQL statement.

NOREGISTER

The optional NOREGISTER attribute indicates the location of the archived redo log file is not to be recorded at the corresponding destination. This setting pertains to remote destinations only. The location of each archived redo log file is always recorded in the primary database control file.

The NOREGISTER attribute is required if the destination is a standby database that is not part of a Data Guard configuration. (For example, the primary database automatically transmits redo data to a standby database only if the standby database is implemented in a Data Guard environment. If a standby database is not established as a part of a Data Guard configuration, you must manually transfer log files using some other means, such as with an operating system copy utility.)

Examples

The following example shows the REGISTER attribute with the LOG_ARCHIVE_DEST_*n* parameter.

```
LOG_ARCHIVE_DEST_5='REGISTER'
```

REOPEN and NOREOPEN

The REOPEN and the NOREOPEN attributes of the LOG_ARCHIVE_DEST_*n* parameter specify the minimum number of seconds before the archiver processes (ARC*n)* or the log writer process (LGWR) should try again to access a previously failed destination. You can turn off the attribute by specifying NOREOPEN.

If neither the REOPEN nor the NOREOPEN attribute is specified, the default is REOPEN.

Category	**REOPEN [=*seconds*]**	**NOREOPEN**
Datatype of the attribute	Numeric	Keyword
Minimum attribute value	0 seconds	Not applicable
Maximum attribute value	Unlimited seconds	Not applicable
Default attribute value	300 seconds	Not applicable
Requires attributes ...	Not applicable	Not applicable
Conflicts with attributes ...	NOREOPEN	REOPEN
Attribute class	ALTER SESSION and ALTER SYSTEM	ALTER SESSION and ALTER SYSTEM
Corresponding V$ARCHIVE_DEST column	REOPEN_SECS	REOPEN_SECS
Related V$ARCHIVE_DEST column	MAX_FAILURE	MAX_FAILURE

REOPEN[=*seconds*]

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 still possible for the Oracle database to overwrite online redo log files even if there is an error. If you specify REOPEN for a MANDATORY destination, log transport services 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.

When you use the REOPEN attribute, note that:

- An archiver process or log writer process reopens a destination only when starting an archival operation from the beginning of the log file and never during a current operation. Archiving always starts copying the log file from the beginning.
- If a value was specified, or the default value was used, for the REOPEN attribute, the archiving process checks if the time of the recorded error plus the REOPEN interval is less than the current time. If it is, the archival operation to that destination is retried.
- You can control the number of times a destination will be retried after a log archiving failure by specifying a value for the MAX_FAILURE=*count* attribute of the LOG_ARCHIVE_DEST_*n* initialization parameter.

NOREOPEN

If you specify NOREOPEN, the failed destination remains disabled until:

- You manually reenable the destination.
- You issue an ALTER SYSTEM SET or an ALTER SESSION SET statement with the REOPEN attribute.
- The instance is restarted.

Examples

The following example shows the REOPEN attribute with the LOG_ARCHIVE_DEST_*n* parameter.

```
LOG_ARCHIVE_DEST_3='SERVICE=stby1 MANDATORY REOPEN=60'
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

SYNC and ASYNC

The SYNC and the ASYNC attributes of the LOG_ARCHIVE_DEST_*n* parameter specify that network I/O is to be done synchronously or asynchronously when 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.

When you specify the LGWR attribute, but you do not specify either the SYNC or ASYNC attribute, the default is SYNC=PARALLEL. When the ARCH attribute is specified for the destination, only the SYNC attribute is valid; an error message is returned if you specify the ARCH and ASYNC attributes together.

Category	**SYNC[=*parallel_option*]**	**ASYNC[=*blocks*]**
Datatype of the attribute	Keyword	Numeric
Minimum attribute value	Not applicable	0
Maximum attribute value	Not applicable	102,400 **Note:** The actual allowable value may be lower, depending on your operating system. Data Guard dynamically adjusts the value down to an appropriate number of blocks, if necessary.
Default attribute value	Not applicable	61,440
Requires attributes ...	Not applicable	LGWR
Conflicts with attributes ...	ASYNC	SYNC, LOCATION, ARCH
Attribute class	ALTER SESSION and ALTER SYSTEM	ALTER SYSTEM
Corresponding V$ARCHIVE_DEST column	TRANSMIT_MODE	TRANSMIT_MODE
Related V$ARCHIVE_DEST column	Not applicable	ASYNC_BLOCKS

Note: If a destination is explicitly configured to use the LGWR process (by specifying the LGWR attribute on the LOG_ARCHIVE_DEST_*n* initialization parameter), but for some reason the log writer process becomes unable to archive to the destination, then log transport services will revert to using the ARC*n* process to complete archival operations using the default (LOG_ARCHIVE_LOCAL_FIRST=TRUE) behavior, even if you specify LOG_ARCHIVE_LOCAL_FIRST=FALSE.

For example, if a standby database problem or a network problem causes the LGWR process to fail, then the ARC*n* process will complete the transmission of redo data to the remote destination. Data Guard minimizes the effect on the primary database as much as possible by archiving to the local destination first to ensure the online redo log files are available to the LGWR process as quickly as possible.

SYNC=PARALLEL SYNC=NOPARALLEL

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 LGWR 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 site before continuing.

If the LGWR process is defined to be the transmitter to multiple standby destinations that use the SYNC attribute, the user has the option of specifying SYNC=PARALLEL or SYNC=NOPARALLEL for each of those destinations.

- If SYNC=NOPARALLEL is used, the LGWR process initiates an I/O to the first destination and waits until it completes before initiating the I/O to the next destination. Specifying SYNC=NOPARALLEL is the same as specifying ASYNC=0.
- If SYNC=PARALLEL is used, the LGWR process submits the network I/O request to the LNS*n* process for that destination and *waits* for an acknowledgment from the LNS*n* process. The LNS*n* process does not write the redo data to a buffer, but immediately transmits it to the standby database, and responds to the waiting LGWR process with status information about the network I/O.

  Specifying SYNC=PARALLEL is useful when you have more than one destination defined with the SYNC attribute. This is because the LGWR process uses a separate LNS*n* process for each destination. Thus, the LGWR issues I/O requests to the LNS*n* process that initiates the network I/O to multiple destinations in parallel. Once the I/O is initiated, the LNS*n* processes wait for all I/O requests to complete before continuing, and the LGWR waits for an acknowledgment from all of the LNS*n* processes. This is, in effect, the same as performing multiple, synchronous I/O requests simultaneously. When you have more than one destination defined with the LGWR and SYNC attributes, the use of `SYNC=PARALLEL` is likely to perform better than `SYNC=NOPARALLEL`. See Figure 5–6 for an illustration of the LNS*n* process in a Data Guard configuration.

Because the `PARALLEL` and `NOPARALLEL` qualifiers only make a difference if multiple destinations are involved, Oracle recommends that all destinations use the same value.

ASYNC[=*blocks*]

The `ASYNC` attribute specifies that network I/O is to be performed asynchronously for the destination. You can optionally specify a block count (from 0 to 102,400) that determines the size of the SGA network buffer to be used. The actual allowable maximum value may be lower, depending on your operating system. Data Guard dynamically adjusts the value down to an appropriate number of blocks, if necessary.

With asynchronous processing, the LGWR process submits the network I/O request to the LNS*n* process for that destination and then LGWR continues processing the next request without waiting for the I/O to complete and without checking the completion status of the I/O. Use of the `ASYNC` attribute allows standby environments to be maintained with little or no performance effect on the primary database.

If the LNS*n* process is slow (for example, due to a slow network), it will result in the LGWR process filling up the ASYNC buffer to its specified capacity, causing an error during asynchronous archival operations. When this happens, the ARC*n* process will eventually transmit the redo data based on the current values of the `REOPEN` and `MAX_FAILURE` attributes for the destination. See Figure 5–6 for an illustration of the LNS*n* process in a Data Guard configuration.

When you use the `ASYNC` attribute, there are several events that cause the network I/O to be initiated:

- If the LGWR request exceeds the current available buffer space, the existing buffer is transmitted to the standby database. The LGWR process waits until sufficient buffer space can be reclaimed.

- A primary database log switch forces any buffered redo data to be transmitted to the standby database before the log switch completes.
- The primary database is shut down normally. An immediate shutdown of the primary database results in the buffered redo data being discarded. A standby database shutdown also causes the buffered redo data for that destination to be discarded.
- The primary database has no redo activity for a period of time. The duration of database inactivity is determined by the system, and you cannot modify it.
- If the rate of redo generation exceeds the runtime network latency and sufficient space is available, then the LGWR request will be buffered. Otherwise, the existing buffer is transmitted to the standby database. The LGWR process waits until sufficient buffer space can be reclaimed.

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 and NOTEMPLATE

The TEMPLATE and the NOTEMPLATE attributes of the LOG_ARCHIVE_DEST_*n* parameter define a directory specification and format template for names of the archived redo log files or standby redo log files at the standby destination. You can specify these attributes in either the primary or standby initialization parameter file, but the attribute applies only to the database role that is archiving.

The TEMPLATE attribute overrides the STANDBY_ARCHIVE_DEST and LOG_ARCHIVE_FORMAT initialization parameter settings at the remote archive destination.

The TEMPLATE and NOTEMPLATE attributes are valid only with remote destinations (that is, destinations that are specified with the SERVICE attribute).

Note: If used on a destination that also specifies the LGWR attribute, rearchiving by the ARC*n* process does not use the TEMPLATE specification. This is important for protected destinations.

There is no default value for this attribute.

Category	TEMPLATE=*filename_template*	NOTEMPLATE
Datatype of the attribute	String value	Not applicable
Minimum attribute value	Not applicable	Not applicable
Maximum attribute value	Not applicable	Not applicable
Default attribute value	Not applicable	Not applicable
Requires attributes ...	SERVICE	Not applicable
Conflicts with attributes ...	NOTEMPLATE, LOCATION, REGISTER=*location_format*	TEMPLATE
Attribute class	ALTER SESSION and ALTER SYSTEM	ALTER SESSION and ALTER SYSTEM
Corresponding V$ARCHIVE_DEST column	REMOTE_TEMPLATE	REMOTE_TEMPLATE

Category	TEMPLATE=*filename_template*	NOTEMPLATE
Related V$ARCHIVE_DEST column	REGISTER	REGISTER

TEMPLATE=*filename_template*

Use the optional TEMPLATE attribute to define a directory specification and format for archive redo log filenames or standby redo log filenames at the standby destination. The definition 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.

The *filename_template* value of the TEMPLATE attribute must contain the %s, %t, and %r directives that are described in Table 12–2.

Table 12–2 Directives for the TEMPLATE Attribute

Directive	Description
%a	Substitute the database activation ID.
%A	Substitute the database activation ID, zero filled.
%d	Substitute the database ID.
%D	Substitute the database ID, zero filled.
%t	Substitute the instance thread number.
%T	Substitute the instance thread number, zero filled.
%s	Substitute the log file sequence number.
%S	Substitute the log file sequence number, zero filled.
%r	Substitute the resetlogs ID.
%R	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.

If you do not specify the TEMPLATE attribute, the setting is the same as REGISTER.

NOTEMPLATE

Use the optional NOTEMPLATE attribute to cancel a previously specified TEMPLATE attribute and allow the filename format template defined by the STANDBY_ARCHIVE_DEST and LOG_ARCHIVE_FORMAT initialization parameters take effect.

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

The VALID_FOR attribute of the LOG_ARCHIVE_DEST_*n* parameter identifies when log 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 online redo log files, standby redo log files, or both are *currently* being archived on the database at this destination

The default value for this attribute is VALID_FOR=(ALL_LOGFILES, ALL_ROLES).

Category	**VALID_FOR=(*redo_log_type, database_role)***
Datatype of the attribute	String value
Minimum attribute value	Not applicable
Maximum attribute value	Not applicable
Default attribute value	VALID_FOR=(ALL LOGFILES, ALL_ROLES) **Note:** 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 10.1.2 for more information.
Requires attributes ...	Not applicable
Conflicts with attributes ...	Not applicable
Attribute class	ALTER SESSION and ALTER SYSTEM
Corresponding V$ARCHIVE_DEST columns	VALID_NOW, VALID_TYPE, and VALID_ROLE
Related V$ARCHIVE_DEST column	Not applicable

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 to 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.

Note: Both the single and plural forms of the keywords are valid. For example, you can specify either PRIMARY_ROLE or PRIMARY_ ROLES, and ONLINE_LOGFILE or ONLINE_LOGFILES.

The following table shows the VALID_FOR attribute values and the roles in which each might be used.

Table 12–3 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)
```

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 log transport services could potentially overwrite the logical standby database's local online redo log files.

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.

The VALID_FOR attribute enables you to set up initialization parameters for the primary and standby roles in the same initialization parameter file. Thus, it is not necessary to maintain separate initialization parameter files when anticipating role reversal in future switchovers or failovers.

Examples

Example 1

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 10.1 for detailed examples of various Data Guard configurations using the VALID_FOR attribute.

VERIFY and NOVERIFY

The VERIFY and NOVERIFY attributes indicate whether or not an archiver (ARC*n*) process should verify the correctness of the contents of a completed archived redo log file.

- VERIFY—thoroughly scans and verifies the completed archived redo log files, either local or remote, for correctness.
- NOVERIFY—indicates that the archived redo log file contents will *not* be verified.

If neither the VERIFY nor the NOVERIFY attribute is specified, the default is NOVERIFY.

Category	**VERIFY**	**NOVERIFY**
Datatype of the attribute	Keyword	Keyword
Minimum attribute value	Not applicable	Not applicable
Maximum attribute value	Not applicable	Not applicable
Default attribute value	Not applicable	Not applicable
Requires attributes ...	Not applicable	Not applicable
Conflicts with attributes ...	NOVERIFY, LGWR	VERIFY
Attribute class	ALTER SYSTEM	ALTER SYSTEM
Corresponding V$ARCHIVE_DEST columns	VERIFY	VERIFY
Related V$ARCHIVE_DEST column	ARCHIVER	ARCHIVER

VERIFY

Use the VERIFY attribute to scan and verify completed archived redo log files, either local or remote, for correctness after successfully completing the archival operation. 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. Consequently, archived redo log file verification is performed *only* when using archiver processes. The use of the VERIFY attribute may have an affect on primary database performance.

NOVERIFY

The default value is `NOVERIFY`, which means that the archived redo log file will not be verified. The `NOVERIFY` attribute indicates that *normal* checksum verification of the archived redo log file will still be performed, but verification of the redo contents will *not* be performed.

12.3 Attribute Compatibility for Archive Destinations

The LOG_ARCHIVE_DEST_*n* initialization parameter has many attributes. Some of these attributes conflict with each other. Some of the attributes require other attributes to be defined. Table 12–4 lists the supported attributes and the requirements associated with each one.

Table 12–4 LOG_ARCHIVE_DEST_n Attribute Compatibility

Attribute	Requires...	Conflicts with...
AFFIRM	Not applicable	NOAFFIRM
NOAFFIRM	Not applicable	AFFIRM
ALTERNATE=*destination*	Not applicable	NOALTERNATE
NOALTERNATE	Not applicable	ALTERNATE
ARCH	Not applicable	LGWR ASYNC NET_TIMEOUT
ASYNC[=*blocks*]	LGWR	SYNC LOCATION ARCH
DB_UNIQUE_NAME	DB_UNIQUE_NAME	NODB_UNIQUE_NAME
NODB_UNIQUE_NAME	Not applicable	Not applicable
DELAY	SERVICE	LOCATION NODELAY
NODELAY	Not applicable	DELAY
DEPENDENCY	SERVICE REGISTER	LOCATION NODEPENDENCY NOREGISTER QUOTA_SIZE QUOTA_USED
NODEPENDENCY	Not applicable	DEPENDENCY
LGWR	Not applicable	ARCH

Table 12–4 (Cont.) LOG_ARCHIVE_DEST_n Attribute Compatibility

Attribute	Requires...	Conflicts with...
LOCATION	Not applicable	SERVICE DEPENDENCY REGISTER=*location_format* NOREGISTER DELAY ASYNC NET_TIMEOUT TEMPLATE QUOTA_SIZE and QUOTA_USED
MANDATORY	Not applicable	OPTIONAL
MAX_FAILURE	REOPEN	NOMAX_FAILURE
NOMAX_FAILURE	Not applicable	MAX_FAILURE
NET_TIMEOUT	LGWR with SYNC=PARALLEL or LGWR with ASYNC > 0	ARCH LOCATION NONET_TIMEOUT LGWR with SYNC=NOPARALLEL LGWR with ASYNC=0
NONET_TIMEOUT	Not applicable	NET_TIMEOUT
OPTIONAL	Not applicable	MANDATORY
QUOTA_SIZE	LOCATION	DEPENDENCY SERVICE NOQUOTA_SIZE LOCATION
NOQUOTA_SIZE	Not applicable	QUOTA_SIZE
QUOTA_USED	LOCATION	DEPENDENCY SERVICE NOQUOTA_USED LOCATION
NOQUOTA_USED	Not applicable	QUOTA_USED
REGISTER	Not applicable	NOALTERNATE NOREGISTER
NOREGISTER	SERVICE	LOCATION REGISTER
REGISTER=*location_ format*	DEPENDENCY	LOCATION NOREGISTER TEMPLATE

Table 12–4 (Cont.) LOG_ARCHIVE_DEST_n Attribute Compatibility

Attribute	Requires...	Conflicts with...
REOPEN	Not applicable	NOREOPEN
NOREOPEN	Not applicable	REOPEN
SERVICE	Not applicable	LOCATION QUOTA_USED QUOTA_SIZE
SYNC[=*parallel_option*]	Not applicable	ASYNC
TEMPLATE	SERVICE	NOTEMPLATE LOCATION REGISTER=*location_format*
NOTEMPLATE	Not applicable	TEMPLATE
VALID_FOR	Not applicable	Not applicable
VERIFY	Not applicable	NOVERIFY, LGWR
NOVERIFY	Not applicable	VERIFY

The LOCATION attribute conflicts with QUOTA_SIZE and QUOTA_USED attributes only when USE_DB_RECOVERY_FILE_DEST is specified on the LOCATION attribute.

13

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 11 for a list of initialization parameters that you can set and dynamically update using the `ALTER SYSTEM SET` or `ALTER SESSION` statements.

13.1 ALTER DATABASE Statements

Table 13–1 describes `ALTER DATABASE` statements that are relevant to Data Guard.

Table 13–1 ALTER DATABASE Statements Used in Data Guard Environments

ALTER DATABASE Statement	Description

`ADD [STANDBY] LOGFILE`
`[THREAD` *integer*`]`
`[GROUP` *integer*`]` *filespec*
	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.
	
	See Section 8.3.5 for an example of this statement.

`ADD [STANDBY] LOGFILE MEMBER` '*filename*' `[REUSE] TO` *logfile-descriptor*	Adds new members to existing online redo log file groups or standby redo log file groups. See Section 5.7.3.2 for an example of this statement.

`[ADD | DROP] SUPPLEMENTAL LOG DATA`
`{PRIMARY KEY|UNIQUE INDEX} COLUMNS`
	This statement is for logical standby databases only.
	
	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 `PRIMARY KEY COLUMNS` or the `UNIQUE INDEX COLUMNS` keyword on this statement.
	
	See Section 4.2.2.1 for an example of this statement.

`COMMIT TO SWITCHOVER TO [PRIMARY]`
`|[[PHYSICAL|LOGICAL] [STANDBY]]`
`[WITH | WITHOUT] SESSION SHUTDOWN]`
`[WAIT | NOWAIT]`
	Performs a switchover to:
	■ Change the current primary database to the standby database role
	■ Change one standby database to the primary database role.
	
	Note: On logical standby databases, you must 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.
	
	See Section 7.2.1 and Section 7.3.1 for examples of this statement.

`CREATE [PHYSICAL | LOGICAL] STANDBY`
`CONTROLFILE AS` '*filename*' `[REUSE]`
	Creates a control file to be used to maintain a physical or a logical standby database. Issue this statement on the primary database.
	
	See Section 3.2.2 and Section 4.2.3.3 for examples of this statement for physical and logical standby databases, respectively.

Table 13–1 (Cont.) ALTER DATABASE Statements Used in Data Guard Environments

ALTER DATABASE Statement	Description
`DROP [STANDBY] LOGFILE` *logfile_descriptor*	Drops all members of an online redo log file group or standby redo log file group. See Section 8.3.5 for an example of this statement.
`DROP [STANDBY] LOGFILE MEMBER` '*filename*'	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 for physical standby databases to prevent inconsistent standby databases. ■ Recommended for logical standby databases to ensure data availability at the standby database. 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.
`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: ■ 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. See Step 5 in Section 4.2.4 for an example of this statement.
`PREPARE TO SWITCHOVER`	This statement is for logical standby databases only. It prepares the primary database and the logical standby database for a switchover by building the LogMiner dictionary *before* 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. See Section 7.3.1 for examples of this statements.

Table 13–1 (Cont.) ALTER DATABASE Statements Used in Data Guard Environments

ALTER DATABASE Statement	Description

`RECOVER MANAGED STANDBY DATABASE [`
`[NO TIMEOUT|TIMEOUT [`*integer*`] ]`
`[NODELAY|DELAY [`*integer*`] ]`
`[DEFAULT DELAY]`
`[DISCONNECT]`
`[NO EXPIRE|EXPIRE [`*integer*`] ]`
`[NEXT [`*integer*`]]`
`[NOPARALLEL|PARALLEL [`*integer*`]]`
`[THROUGH {ALL|NEXT|LAST SWITCHOVER]`
`[THROUGH [THREAD `*n*`] SEQUENCE `*n*`]`
`[ALL ARCHIVELOG]`
`[FINISH [SKIP[STANDBY LOGFILE]`
`[NOWAIT|WAIT]]`
`[UNTIL CHANGE `*scn*`]`
`[USING CURRENT LOGFILE] ]`
	This statement is for physical standby databases only.
	
	Use this statement to start and control log apply services for physical standby databases. You can use the `RECOVER MANAGED STANDBY DATABASE` clause on a physical standby database that is mounted, open, or closed. This clause provides many options to help you control Redo Apply.
	
	See Step 3 in Section 3.2.6 and Section 6.3 for examples of this statement.

`RECOVER MANAGED STANDBY DATABASE CANCEL`
`[[NOWAIT]|[WAIT]|[IMMEDIATE] ]`
	This statement cancels Redo Apply on a physical standby database.

`REGISTER [OR REPLACE]` `[PHYSICAL|LOGICAL] LOGFILE `*filespec*	Allows the registration of manually archived redo log files. See Section 5.8.4 for an example of this statement.
`RESET DATABASE TO INCARNATION `*integer*	Resets the target recovery incarnation for the database from the current incarnation to the prior incarnation.

`SET STANDBY DATABASE TO MAXIMIZE`
`{PROTECTION|AVAILABILITY|PERFORMANCE}`
	Issue this statement on any primary database that is mounted but not opened. Specifies one of the three data protection modes for the Data Guard configuration. All three modes provide a high degree of data protection, but they differ in terms of the effect that each protection mode has on the availability and performance of the primary database. The
	
	See Section 5.6.3 for an example of this statement.

Table 13–1 (Cont.) ALTER DATABASE Statements Used in Data Guard Environments

ALTER DATABASE Statement	Description
`START LOGICAL STANDBY APPLY INITIAL [`*scn-value*`] ] [NEW PRIMARY` *dblink*`]`	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 [SKIP [STANDBY LOGFILE]]`	Performs a failover in which the primary database is removed from the Data Guard environment and one standby database assumes the primary database role. 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: ■ For physical standby databases, use the `ALTER DATABASE RECOVER MANAGED STANDBY DATABASE` statement with the `FINISH` or `FINISH SKIP` keywords, which 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.

13.2 ALTER SESSION Statements

Table 13–2 describes an `ALTER SESSION` statement that is relevant to Data Guard.

Table 13–2 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.

14 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 14–1 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 14–1 Views That Are Pertinent to Data Guard Configurations

View	Database	Description
DBA_LOGSTDBY_EVENTS	Logical only	Contains information about the activity of the logical standby database system. It can be used to determine the cause of failures that occur when SQL Apply is applying redo to logical standby databases.
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_PROGRESS	Logical only	Describes the progress of SQL Apply on the logical standby database.
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 datatypes. Use this view when you are preparing to create a logical standby database.

Table 14–1 (Cont.) Views That Are Pertinent to Data Guard Configurations

View	Database	Description
V$ARCHIVE_DEST	Primary, physical, and logical	Describes, for the current instance, all of the destinations in the Data Guard configuration, including each destination's current value, mode, and status. **Note:** The information in this view does not persist across an instance shutdown.
V$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.
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$DATABASE	Primary, physical, and logical	Provides database information from the control file.
V$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_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.

Table 14–1 (Cont.) Views That Are Pertinent to Data Guard Configurations

View	Database	Description
V$LOG_HISTORY	Primary, physical, and logical	Contains log history information from the control file.
V$LOGSTDBY	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$LOGSTDBY_STATS	Logical only	Displays LogMiner statistics, current state, and status information for a logical standby database during SQL Apply. If SQL Apply is not running, the values for the statistics are cleared.
V$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:

A

Troubleshooting Data Guard

This appendix provides help troubleshooting a standby database. This appendix contains the following sections:

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/data files
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, log 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.
- 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, log 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 `NOREOPEN` attribute was specified or the `MAX_FAILURE` attribute threshold was exceeded, log 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 *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
- Switchover Fails Because Redo Data Is Not Applied After the 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 14 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 JOB_QUEUE_PROCESSES 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 AQ_TM_PROCESSES 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 agentctl stop 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. 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 Switchover Fails Because Redo Data Is Not Applied After the Switchover

The archived redo log files are not applied to the 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 primary site and the `listener.ora` file at the 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 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 E 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 `DBA_LOGSTDBY_EVENTS` 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.

Table A–2 (Cont.) Fixing Typical SQL Apply Errors

If...	Then...
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 `ALTER DATABASE START LOGICAL STANDBY APPLY` 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 `DBMS_LOGSTDBY.SKIP_TRANSACTION` procedure to ensure the incorrect statement is ignored the next time SQL Apply is run. Then, restart SQL Apply using the `ALTER DATABASE START LOGICAL STANDBY APPLY` statement.
An error occurred because skip parameters were incorrectly set up, such as specifying that all DML for a given table be skipped but `CREATE`, `ALTER`, and `DROP TABLE` statements were not specified to be skipped	Issue the `DBMS_LOGSTDBY.SKIP('TABLE','schema_name','table_name',null)` procedure, then restart SQL Apply.

See Chapter 14 for information about querying the `DBA_LOGSTDBY_EVENTS` view to determine the cause of failures.

A.6 Network Tuning for Redo Data Transmission

The process of transmitting redo data involves reading a buffer from the online redo log file and writing it to the archived redo log file location. When the destination is remote, the buffer is written to the archived redo log file location over the network using Oracle Net services.

The default archived redo log file buffer size is 1 megabyte. The default transfer buffer size for Oracle Net is 2 kilobytes. Therefore, the archived redo log file buffer is divided into units of approximately 2 kilobytes for transmission. These units could get further divided depending on the maximum transmission unit (MTU) of the underlying network interface.

The Oracle Net parameter that controls the transport size is **session data unit (SDU).** This parameter can be adjusted to reduce the number of network packets that are transmitted. This parameter allows a range of 512 bytes to 32 kilobytes.

For optimal performance, set the Oracle Net `SDU` parameter to 32 kilobytes for the associated `SERVICE` destination parameter.

The following example shows a database initialization parameter file segment that defines a remote destination `netserv`:

```
LOG_ARCHIVE_DEST_3='SERVICE=netserv'
```

```
SERVICE_NAMES=srvc
```

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)(ORACLE_HOME=/oracle)))
```

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 high-latency or high-bandwidth connections, you can improve performance by increasing the TCP send and receive window sizes.

If high-speed WAN links are used to connect the sites in a Data Guard configuration, network throughput can often be substantially improved 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 Net Services Administrator's Guide*.

A.7 Managing Data Guard Network Timeout

For any given Oracle Data Guard network connection, there are two processes communicating with each other. When the network connection is unexpectedly broken, how these processes react differs greatly. This is a discussion of what actually occurs when a network connection is broken, and how it affects the Data Guard environment and configuration. This discussion applies to both physical and logical standby databases.

Data Guard uses a peer-to-peer connection protocol, whereby a primary database process, whether it is the log writer process (LGWR) or archiver processes (ARC*n*), establishes a network connection to the standby database. As a result of the network connection request, the listener on the standby site creates a separate process on the standby database called the Remote File Server (RFS) process. The RFS process uses network messages from the primary database; it reads from the network and sends an acknowledgment message back to the primary database when it is done processing the request.

During normal Data Guard operations, when redo data is transmitted from the primary database to the standby database, network messages are initiated from the primary database (the network client), and always acknowledged by the standby database (the network server). In this case, the LGWR and ARCH processes are the network clients, and the RFS process is the network server.

Consider the simple scenario where the network between the primary and standby systems is disconnected. When the LGWR process attempts to send a new message to the RFS process over this connection, the LGWR process receives an error from Oracle Net, after a TCP timeout, indicating that the connection is broken. In this way, the LGWR is able to establish that network connectivity is lost, and take corrective action. The Data Guard attributes `[NO]MAX_FAILURE`, `[NO]REOPEN` and `[NO]NET_TIMEOUT`, which are options for the `LOG_ARCHIVE_DEST_`*n* parameter, provide LGWR with the desired flexibility to control the timeout intervals and number of retries associated with a network connection that is not responding.

In contrast to the LGWR process, the RFS process on the standby database is always synchronously waiting for a new message to arrive from the primary database. The RFS process that is doing the network read operation is blocked until some data arrives, or until the underlying network software determines the connection is no longer valid.

Oracle Net periodically sends a network probe to verify a client/server connection is still active. This ensures connections are not left open indefinitely due to an abnormal client termination. If the probe finds a broken connection, it returns an error that causes the RFS process to exit.

You can use the Oracle Net `SQLNET.EXPIRE_TIME` parameter to specify the time interval, expressed in minutes, when to send a probe to verify the network session is active. Setting this parameter to a small value allows for more timely detections of broken connections. Connections that do not respond to this probe signal are disconnected. This parameter should be set up for the standby database, as well as the primary database, to prepare it for future switchover scenarios.

Limitations on using this feature are:

- Though very small, a probe packet generates additional traffic. However, compared to the network traffic generated by Data Guard that is based on the primary database workload, this additional packet traffic is insignificant.
- Depending on which operating system is in use, the server might need to perform additional processing to distinguish the connection-probing event from other events that occur. This can affect network performance.

Once the RFS process receives notification of the broken network connection, it will terminate itself. However, until such time as the RFS process terminates itself, it will retain lock information on the archived redo log file on the standby site, or the standby redo log file, whose redo data was being received from the primary database. During this interval, no new RFS processes can receive redo data from the primary database for the same archived redo log file (or the standby redo log file).

Oracle recommends setting the Oracle Net `SQLNET.EXPIRE_TIME` parameter to 1 minute. This is a reasonable value for most systems, and setting the parameter to a small value does not significantly impact production systems.

Once the network problem is resolved, and the primary database processes are again able to establish network connections to the standby database, a new RFS process will automatically be started on the standby database for each new network connection. These new RFS processes will resume the reception of redo data from the primary database.

A.8 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 you should set is 1 MB.

A.9 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 current standby redo log file on each standby database exactly matches the size of the current online redo log file 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 primary database will change to maximum performance mode if it is operating in maximum availability mode.
- 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 and 200K, respectively, then the standby database should have 4 standby redo log groups with log file sizes of 100K and 200K.

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 5.6.2, "Configuring Standby Redo Log Files" for more information.

B 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

B.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:

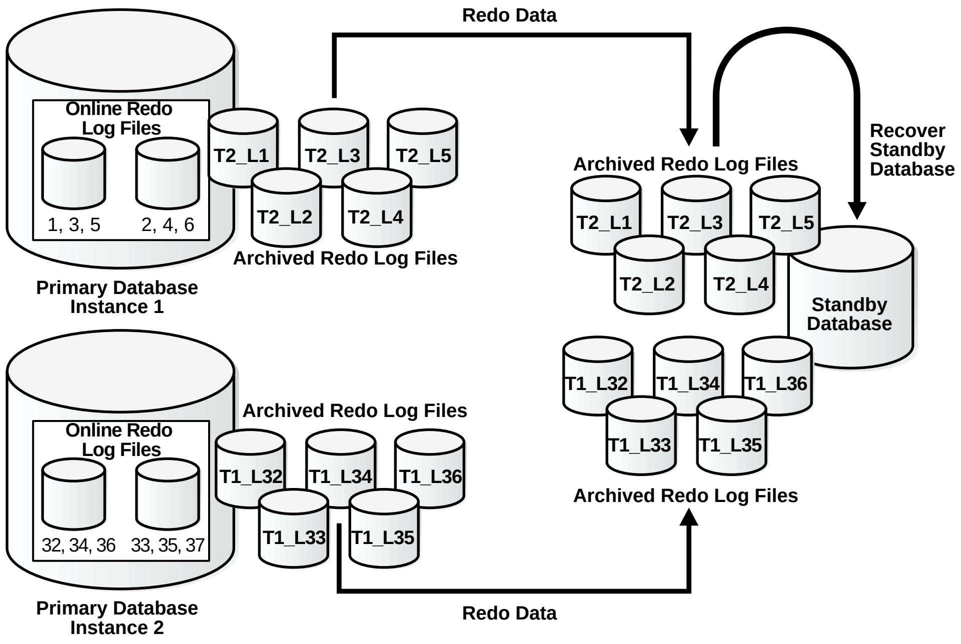
Instance Combinations	Single-Instance Standby Database	Multi-Instance Standby Database
Single-instance primary database	Yes	Yes
Multi-instance primary database	Yes	Yes

In each scenario, each instance of the primary database transmits its own redo data to archived redo log files on the standby database.

B.1.1 Setting Up a Multi-Instance Primary with a Single-Instance Standby

Figure B–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 B–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.

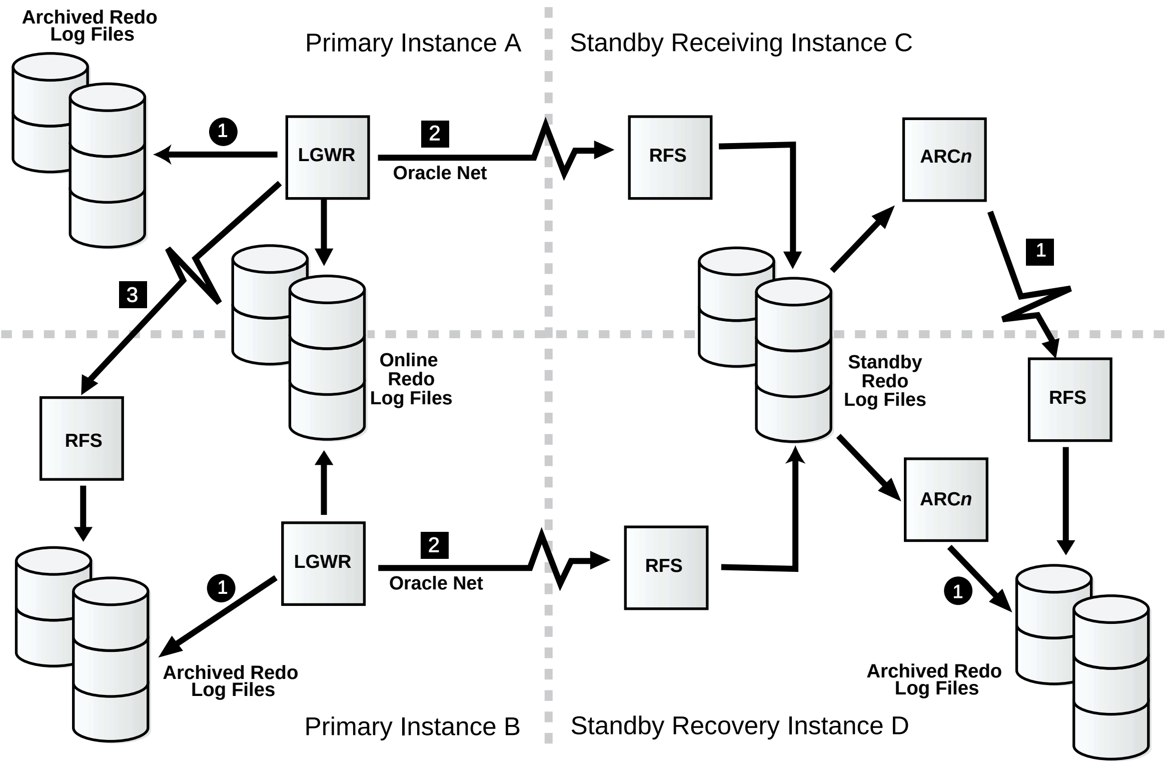
To set up a single instance standby database

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.

B.1.2 Setting Up a Multi-Instance Primary with a Multi-Instance Standby

Figure B–2 shows a configuration where the primary and standby databases are in a Real Application Clusters environment. This enables you to separate the log transport services processing from the log apply services processing on the standby database, thereby improving overall primary and standby database performance.

Figure B–2 Standby Database in Real Application Clusters



In Figure B–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 (ARC*n*) 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 log 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 5.6.2 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 log 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.

B.1.3 Setting Up a Cross-Instance Archival Database Environment

In a cross-instance archival environment, each instance directs its archived redo log files to a single instance of the cluster. This instance is called the **recovery instance**. This instance typically has a tape drive available for RMAN backup and restore support. Example B–1 shows how to set up the `LOG_ARCHIVE_DEST_`*n* initialization parameter for archiving redo data across instances. Execute these statements on all instances except the recovery instance.

Example B–1 Setting Destinations for Cross-Instance Archiving

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_1 = 'LOCATION=archivelog MANDATORY REOPEN=120';
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_1 = enable;
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_2 = 'SERVICE=prmy1 MANDATORY REOPEN=300';
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2 = enable;
```

Destination 1 is the repository containing the local archived redo log files required for instance recovery. This is a mandatory destination. Because the expected cause of failure is lack of adequate disk space, the retry interval is 2 minutes. This should be adequate to allow the DBA to purge unnecessary archived redo log files. Notification of destination failure is accomplished by manually searching the primary database alert log.

Destination 2 is the recovery instance database where RMAN is used to back up the archived redo log files from local disk storage to tape. This is a mandatory destination, with a reconnection threshold of 5 minutes. This is the time needed to fix any network-related failures. Notification of destination failure is accomplished by manually searching the primary or standby database alert log.

Cross-instance archiving is available using the ARC*n* process only. Using the LGWR process for cross-instance archiving results in the RFS process failing, and the archive log destination being placed in the Error state.

B.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

B.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 B–1.

Table B–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.

B.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.

B.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 data integrity of the data that has been transmitted to that destination and can be recovered).

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 standby destination, the instance loses connectivity and will be shut down. Other destinations 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.

B.2.4 Role Transitions

This section contains the following topics:

- Switchovers
- Failovers

B.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 always manually add redo log files when configuring RAW devices for physical standby databases.

B.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.

B.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

B.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
```

B.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 participating 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 5.6.3 (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.

C

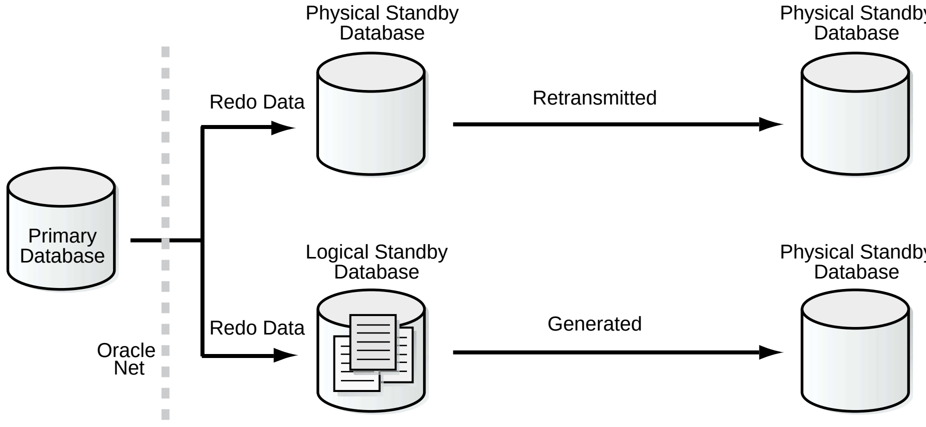
Cascaded Redo Log Destinations

To reduce the load on your primary system, you can implement **cascaded redo log 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 C–1 shows cascaded redo log destinations to physical and logical standby databases.

Figure C–1 Cascaded Redo Log 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 `LOG_ARCHIVE_DEST_`*n* and `LOG_ARCHIVE_DEST_STATE_`*n* parameters 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 Redo Log Destinations
- Role Transitions with Cascaded Redo Log Destinations
- Examples of Cascaded Redo Log Destinations

C.1 Configuring Cascaded Redo Log Destinations

The following sections describe how to set up a Data Guard configuration to use cascaded redo log destinations:

- Configuring Cascaded Redo Log Destinations for Physical Standby Databases
- Configuring Cascaded Redo Log Destinations for Logical Standby Databases

C.1.1 Configuring Cascaded Redo Log 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 to use the `LGWR` transport method. Use either `SYNC` or `ASYNC` network protocols depending on your requirements.
- On the receiving physical standby database, define sufficient 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. Remember, as part of the original setup of the physical standby database, you should have defined a local archive destination that will be used for local archiving when the physical standby database transitions to the primary role. For example, you might define the `LOG_ARCHIVE_DEST_1` initialization parameter to be the `'LOCATION=/physical1/arch'` location. When the physical standby database switches roles, any archived redo log files will be put into that directory with the same format that you defined with the `LOG_ARCHIVE_FORMAT` initialization parameter. This local archiving destination can be the same as the one defined in the parameter `STANDBY_ARCHIVE_DEST`, but this is not required.

A side effect of this configuration is that the archiver processes on the standby database will now try to send the redo data not only to the cascading end points but also to the other standby databases and the primary database if they are defined and enabled. This is not a problem, because the receiving database will either reject it if it is the primary database or a standby database that has already received the same redo data successfully. If the destination is another standby database and it has not received the log file successfully, then this acts as an active gap resolution. You can avoid this by setting the state to `DEFER` for any destinations not involved in the cascade. However, you will have to remember to enable them again if you do a switchover or failover operation.

If you want to have one initialization parameter file to handle both the cascaded redo log destinations and the original primary and standby destinations, define the destinations for the primary database and other standby databases as well as the cascading standby databases. However, the total remote destinations still cannot exceed 10, including the local archiving destination.

C.1.2 Configuring Cascaded Redo Log 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. You can use any transport mode (`LGWR` or `ARCH`) and network protocol (`SYNC` or `ASYNC`). If you use the `LGWR` transport mode, you can optionally use standby redo log files on your standby databases.

C.2 Role Transitions with Cascaded Redo Log 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.

C.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 redo log 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.

C.2.2 Standby Databases Receiving Redo Data from a Logical Standby Database

Any standby database that receives redo datacascaded 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.

C.3 Examples of Cascaded Redo Log Destinations

The following scenarios demonstrate configuration options and uses for cascaded redo log destinations.

C.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.

C.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:

- The physical standby database is receiving its redo data from the log writer of the primary database.
- Standby redo log files are defined and 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. If the `LGWR` network transport is not being used, you can dynamically set log transport services on the primary database, and the primary database will start using the log writer at the next log switch.

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.

C.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 redo log 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 redo log 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. 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.

C.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 or a WAN. 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 log 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.

C.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 smallerlocal system. A distribution repository is contains only the standby control file and the standby database archived redo log files, not the data files. 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.

D Creating a Physical Standby Database with Recovery Manager

This appendix describes how to use Oracle Recovery Manager to create a physical standby database. This appendix contains the following topics:

D.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 database.
- RMAN automates renaming of files, or 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 physical standby database in Chapter 3 *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

D.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 D–1.

Table D–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 *Oracle Database Backup and Recovery Basics*.
Create a backup of the primary control file that is usable as a standby control file (if you do not have one).	See Section D.1.2, "Creating the Standby Control File with RMAN".
Choose filenames for the standby datafiles.	See Section D.1.3, "Naming the Standby Database Datafiles When Using RMAN".
Choose filenames for the standby database archived redo log files and standby redo log files.	See Section D.1.4, "Naming the Standby Database Log Files When Using RMAN".

In addition to the RMAN tasks described in Table D–1, you must also perform the following additional tasks to set up your standby database:

- Set all necessary initialization parameters in the primary initialization parameter file.
- 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.

D.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 3 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';
```

D.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 filenames the same as the primary filenames (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 filenames in the standby control file in multiple ways, a method for prioritizing settings is necessary. Table D–2 specifies the hierarchy for the naming of datafiles in the standby database.

Table D–2   Order of Precedence for Naming Datafiles in Standby Database

	Method of Standby Datafile Naming	**Requirement**
1	Issue `SET NEWNAME` command.	You must issue this command in the `RUN` block for the creation of the standby database.
2	Use the `DB_FILE_NAME_CONVERT` option of the RMAN `DUPLICATE` command.	None.
3	Issue `CONFIGURE AUXNAME` command.	You must be connected to a recovery catalog, and an `AUXNAME` that is not `NULL` must be stored in the catalog for the datafile.

Table D–2 (Cont.) Order of Precedence for Naming Datafiles in Standby Database

	Method of Standby Datafile Naming	Requirement
4	Datafile filename as currently specified in the standby control file. The standby filename is identical to the primary filename or is named with the DB_FILE_NAME_CONVERT parameter.	If the filename is different, then the DB_FILE_NAME_CONVERT parameter must be set in the standby initialization parameter file. If the filename is the same, then you must specify the NOFILENAMECHECK clause of the DUPLICATE command.

See *Oracle Database Reference* for more information about how to use DB_FILE_NAME_CONVERT to name standby files.

D.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 filenames for the log files, as specified in the standby control file. If the log filenames on the standby must be different from the primary filenames, then one option is to specify filenames for the redo logs by setting LOG_FILE_NAME_CONVERT in the standby initialization parameter file.

Note these restrictions when specifying filenames 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 filenames for the redo log files.
- If you want the redo log filenames on the standby database to be the same as the primary redo log filenames, 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.

D.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 attempt 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, run the Recovery Manager `DUPLICATE ... FOR STANDBY` command to create the standby database out of 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 whether or not you specify that RMAN should recover the standby database after creating it.

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.

D.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.

D.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 D.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 D.7.

See *Oracle Database Recovery Manager Reference* for the list of DUPLICATE restrictions for creating a standby database with RMAN.

D.3 Setting Up the Standby Instance

No matter which standby creation scenario you choose, you must first start the standby instance and then connect RMAN to this instance. The details of this procedure vary depending on whether or not the standby and primary sites have a different directory structure.

To start the standby instance:

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 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:

   - Initialization parameters 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
   ```

D.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 filenames for the standby datafiles. The primary and standby datafiles and log files have the same filenames.

D.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 D.3. 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 you do not have automatic channels configured, then manually 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;
   ```

D.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 D.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.

D.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 filenames 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.

D.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 3.1.3 for examples of how to use the DB_FILE_NAME_CONVERT and LOG_FILE_NAME_CONVERT parameters to name standby database files.

D.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 D.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.

D.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 D.4.2.

D.5.2 Naming Standby Database Files with SET NEWNAME

In this procedure, you use SET NEWNAME commands to name the standby datafiles.

D.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 D.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 filenames 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 filenames 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;
}
```

D.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 D.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 D.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 filenames 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 filenames 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.

D.5.3 Naming Standby Database Files with CONFIGURE AUXNAME

In this procedure, you use CONFIGURE AUXNAME commands to name the standby datafiles.

D.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 D.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;
```

D.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 D.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 D.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;
```

D.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 D.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 filenames 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.

D.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

D.7.1 Overview

The main restriction when using RMAN image copies to create the standby datafiles is that the image copy filenames 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 filename 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 NT).

The filename of the image copy on the standby host must be the same as the filename 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 D–3 illustrates two scenarios for using NFS to create a standby database with one datafile.

Table D–3 Using Image Copies to Create a Standby Database: Scenario

NFS Mount Point	Primary Datafile Filename	Image Copy Filename	Standby Datafile Filename	Procedure
`/data` (same on both hosts)	`/oracle/dbs/df1.f`	`/data/df1.f`	`/data/df1.f` (same path name as image copy)	See Section D.7.2, "When Copies and Datafiles Use the Same Names"
`/data` (same on both hosts)	`/oracle/dbs/df1.f`	`/data/df1.f`	`/oracle/sb/df1.f` (different path name from image copy)	See Section D.7.3, "When Copies and Datafiles Use Different Names"

Table D–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 filenames 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 filename `/oracle/dbs/df1.f` to the standby filename `/data/df1.f`. Then, copy the files to the standby host, and mount the standby database.

In the second scenario, you use different filenames 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.

D.7.2 When Copies and Datafiles Use the Same Names

This procedure assumes that you are using the same filenames 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 filenames are translated from the primary datafile filenames. 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;
   ```

D.7.3 When Copies and Datafiles Use Different Names

This procedure assumes that you use different filenames for the standby datafiles and the image copies of the primary datafiles.

D.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 DB_FILE_NAME_CONVERT in the standby initialization parameter file so that standby datafile filenames are translated from the primary datafile filenames, 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
   ```

D.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 filenames are translated from the primary datafile filenames, 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 D.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.

D.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 filename 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 '/standy/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.

E 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 AR*Cn*, LGWR, and foreground processes on the primary database, and the RFS and FAL server processes on the standby database.

E.1 LOG_ARCHIVE_TRACE Initialization Parameter

To see the archiving progression 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 (ARC*n* 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 ARC*n* 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.

E.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  ?/rdbms/log
```

E.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 that is performing Redo Apply or is in read-only mode, 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 E.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 ARC*n* 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;
```

E.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

Level	Meaning
8	Tracks archive log destination activity
16	Tracks detailed archive log destination activity
32	Tracks archive log destination parameter modifications
64	Tracks ARC*n* process state activity
128	Tracks FAL server process activity
256	Supported in a future release
512	Tracks asynchronous LGWR activity
1024	Tracks the RFS physical client
2048	Tracks the ARC*n* or RFS heartbeat
4096	Tracks real-time apply activity

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/d1arc1_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/d1arc1_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/d1arc1_387.log
( 4)    ARC0: FINISH
( 2)    ARC0: Archival success destination 2 : 'standby1'
( 2)    ARC0: Archival success destination 1 : '/oracle/dbs/d1arc1_387.log'
( 4)    ARC0: COMPLETE, all destinations archived
(16)    ARC0: ArchivedLog entry added: /oracle/dbs/d1arc1_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 KCRRNOARCH->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
```

F

Sample Disaster Recovery ReadMe File

In a multiple standby database configuration, you cannot assume that the database administrator (DBA) who set up the multiple standby database configuration is available to decide which standby database to fail over to in the event of a disaster. 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.

Example F–1 shows the kind of information that the person who is making the decision would need when deciding which standby database should be the target of the failover.

A ReadMe file is created and maintained by the DBA and should describe how to:

- Log on to the local database as a DBA
- Log on to each system where the standby databases are located

  There might be firewalls between systems. The ReadMe file should include instructions for going through the firewalls.
- Log on to other 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

Example F–1 Sample Disaster Recovery ReadMe File

```
----------------Standby Database Disaster Recovery ReadMe File----------------

Warning:
```

```
********************************************************************************
Perform the steps in this procedure only if you are responsible for failing over
to a standby database after the primary database fails.

If you perform the steps outlined in this file unnecessarily, you might corrupt
the entire database system.
********************************************************************************

Multiple Standby Database Configuration:

No.     Location       Type     IP Address
--- --------------- --------- --------------
 1   San Francisco   Primary   128.1.124.25
 2   San Francisco   Standby   128.1.124.157
 3   Boston          Standby   136.132.1.55
 4   Los Angeles     Standby   145.23.82.16
 5   San Francisco   Standby   128.1.135.24

You are in system No. 3, which is located in Boston.

Perform the following steps to fail over to the most up-to-date and available
standby database:

1. Log on to the local standby database as a DBA.

    a)  Log on with the following user name and password:

                username: Standby3
                password: zkc722Khn

    b)  Invoke SQL*Plus as follows:

        % sqlplus

    c)  Connect as the DBA as follows:

        CONNECT sys/s23LsdIc AS SYSDBA

2.  Connect to as many remote systems as possible. You can connect to a maximum
    of four systems. System 4 does not have a firewall, so you can connect to it
    directly. Systems 1, 2, and 5 share the same firewall host.  You need to go
    to the firewall host first and then connect to each system.  The IP address
    for the firewall host is 128.1.1.100.  Use the following user name and
    password:
               username: Disaster
```

```
           password: 82lhsIW32

3.  Log on to as many remote systems as possible with the following user names
    and passwords:

    Login information:

    No.     Location     IP Address    username   password
    --- --------------- ------------ ---------- ----------
    1   San Francisco   128.1.124.25   Oracle9i   sdd290Ec
    2   San Francisco   128.1.124.157  Standby2   ei23nJHb
    3                    (L o c a l)
    4   Los Angeles     145.23.82.16   Standby4   23HHoe2a
    5   San Francisco   128.1.135.24   Standby5   snc#$dnc

4.  Invoke SQL*Plus on each remote system you are able to log on to as follows:

    % sqlplus

5.  Connect to each remote database as follows:

    CONNECT sys/password AS SYSDBA

    The DBA passwords for each location are:

    No.     Location      Password
    --- --------------- -----------
    1   San Francisco     x2dwlsd91
    2   San Francisco     a239s1DAq
    3           (L o c a l)
    4   Los Angeles       owKL(@as23
    5   San Francisco     sad_KS13x

6.  If you are able to log on to System 1, invoke SQL*Plus and execute the
    following statements:

    SQL> SHUTDOWN IMMEDIATE;
    SQL> STARTUP PFILE=PRMYinit.ora;

    Note: If you are able to execute the STARTUP statement successfully, the
          primary database has not been damaged.  Do not continue with this
          procedure.

7.  Execute the following SQL statements on each standby database (including the
    one on this system) that you were able to connect to:
```

```
    SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE CANCEL;
    SQL> SELECT THREAD#, MAX(SEQUENCE#) FROM V$LOG_HISTORY GROUP BY THREAD#;

    Compare the query results of each standby database. Fail over to the
    standby database with the largest sequence number.

8.  Fail over to the standby database with the largest sequence number.

    On the standby database with the largest sequence number, invoke SQL*Plus
    and execute the following SQL statements:

    SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE
      2> DISCONNECT FROM SESSION;
    SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE FINISH;
    SQL> ALTER DATABASE COMMIT TO SWITCHOVER TO PRIMARY;
    SQL> SHUTDOWN IMMEDIATE;
    SQL> STARTUP PFILE=Failover.ora;

9.  Update the other standby databases with the new primary database information
and ensure the log transport and log apply services are working correctly.

------------End of Standby Database Disaster Recovery ReadMe File-------------
```

Index