Oracle® Data Guard

Concepts and Administration

11*g* Release 1 (11.1)

B28294-03

August 2008

ORACLE®

Oracle Data Guard Concepts and Administration, 11*g* Release 1 (11.1)

B28294-03

Primary Author: Kathy Rich

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

Contents

9 Managing Physical and Snapshot Standby Databases

10 Managing a Logical Standby Database

11 Using RMAN to Back Up and Restore Files

12 Using SQL Apply to Upgrade the Oracle Database

13 Data Guard Scenarios

D Data Guard and Oracle Real Application Clusters

E Cascaded Destinations

F Creating a Standby Database with Recovery Manager

G Setting Archive Tracing

List of Examples

List of Figures

List of Tables

Preface

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

Audience

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

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

Documentation Accessibility

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

Accessibility of Code Examples in Documentation

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

Accessibility of Links to External Web Sites in Documentation

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

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, 7 days a week. For TTY support, call 800.446.2398. Outside the United States, call +1.407.458.2479.

Related Documents

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

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

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

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

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

Conventions

The following text conventions are used in this document:

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

What's New in Oracle Data Guard?

This preface describes the new features added to Oracle Data Guard in release 11.1 and provides links to additional information. The features and enhancements described in this preface were added to Oracle Data Guard in 11*g* Release 1 (11.1). The new features are described under the following main areas:

- New Features Common to Redo Apply and SQL Apply
- New Features Specific to Redo Apply and Physical Standby Databases
- New Features Specific to SQL Apply and Logical Standby Databases

New Features Common to Redo Apply and SQL Apply

The following enhancements to Oracle Data Guard in 11*g* Release 1 (11.1) improve ease-of-use, manageability, performance, and include innovations that improve disaster-recovery capabilities:

- **Compression of redo traffic over the network in a Data Guard configuration**

  This feature improves redo transport performance when resolving redo gaps by compressing redo before it is transmitted over the network.

  See Also: "COMPRESSION" attribute on page 15-5

- **Redo transport response time histogram**

  The `V$REDO_DEST_RESP_HISTOGRAM` dynamic performance view contains a histogram of response times for each `SYNC` redo transport destination. The data in this view can be used to assist in the determination of an appropriate value for the `LOG_ARCHIVE_DEST_`*n* NET_TIMEOUT attribute.

  See Also: "NET_TIMEOUT" attribute on page 15-17.

- **Faster role transitions**
- **Strong authentication for redo transport network sessions**

  Redo transport network sessions can now be authenticated using SSL. This provides strong authentication and makes the use of remote login password files optional in a Data Guard configuration.

- **Simplified Data Guard management interface**

  The SQL statements and initialization parameters used to manage a Data Guard configuration have been simplified through the deprecation of redundant SQL clauses and initialization parameters.

See Also:

- Chapter 16, "SQL Statements Relevant to Data Guard" for information about which statements have had clauses deprecated
- *Oracle Database SQL Language Reference* for more information about deprecated clauses relevant to the SQL statements discussed in Chapter 16
- *Oracle Database Reference* for information about deprecated attributes of the `LOG_ARCHIVE_DEST_n` parameter

- **Enhancements around `DB_UNIQUE_NAME`**

  You can now find the `DB_UNIQUE_NAME` of the primary database from the standby database by querying the new `PRIMARY_DB_UNIQUE_NAME` column in the `V$DATABASE` view. Also, Oracle Data Guard release 11*g* ensures each database's `DB_UNIQUE_NAME` is different. After upgrading to 11g, any databases with the same `DB_UNIQUE_NAME` will not be able to communicate with each other.

- **Use of physical standby database for rolling upgrades**

  A physical standby database can now take advantage of the rolling upgrade feature provided by a logical standby. Through the use of the new `KEEP IDENTITY` clause option to the SQL `ALTER DATABASE RECOVER TO LOGICAL STANDBY` statement, a physical standby database can be temporarily converted into a logical standby database for the rolling upgrade, and then reverted back to the original configuration of a primary database and a physical standby database when the upgrade is done.

  See Also: Chapter 12, "Using SQL Apply to Upgrade the Oracle Database"

- **Heterogeneous Data Guard Configuration**

  This feature allows a mix of Linux and Windows primary and standby databases in the same Data Guard configuration.

  Note: Also, see the "What's New" preface in *Oracle Data Guard Broker* for details about the enhanced broker-based management framework, including:

  - Fast-start failover for maximum performance mode in a Data Guard configuration
  - User-configurable events to trigger fast-start failover in a Data Guard configuration

  ---

- **The ARCH redo transport mode has been deprecated**

  The ARCH redo transport mode has been deprecated and will be desupported in a future release. Oracle recommends that you switch to the ASYNC transport mode if you are currently using the ARCH transport mode. The ASYNC transport mode is superior to the ARCH transport mode in all respects, and is the new default transport mode.

New Features Specific to Redo Apply and Physical Standby Databases

The following list summarizes the new features that are specific to Redo Apply and physical standby databases in Oracle Database 11*g* Release 1 (11.1):

- **Real-time query capability of physical standby**

  This feature makes it possible to query a physical standby database while Redo Apply is active.

  > **See Also:** Section 9.2, "Opening a Physical Standby Database" on page 9-2 for more information about how an open physical standby database can continue to receive and apply redo data from a primary database

- **Snapshot standby**

  A snapshot standby database is new type of updatable standby database that provides full data protection for a primary database.

  > **See Also:** Section 9.7, "Managing a Snapshot Standby Database"

- **Lost-write detection using a physical standby**

  A "lost write" is a serious form of data corruption that can adversely impact a database. It occurs when an I/O subsystem acknowledges the completion of a block write in the database, while in fact the write did not occur in the persistent storage. This feature allows a physical standby database to detect lost writes to a primary or physical standby database.

  > **See Also:** Section 13.6, "Recovering From Lost-Write Errors on a Primary Database" for an example of lost-write recovery, and *Oracle Database Backup and Recovery User's Guide* for information about enabling lost-write detection

- **Improved Integration with RMAN**

  A number of enhancements in RMAN help to simplify backup and recovery operations across all primary and physical standby databases, when using a catalog. Also, you can use the RMAN `DUPLICATE` command to create a physical standby database over the network without a need for pre-existing database backups.

  > **See Also:**
  >
  > - Section 3.2, "Step-by-Step Instructions for Creating a Physical Standby Database"
  > - Chapter 11, "Using RMAN to Back Up and Restore Files"
  > - Appendix F, "Creating a Standby Database with Recovery Manager"

New Features Specific to SQL Apply and Logical Standby Databases

The following list summarizes the new features for SQL Apply and logical standby databases in Oracle Database 11*g* Release 1 (11.1):

- **Support for additional object datatypes and PL/SQL package support**
  - XML stored as `CLOB`

See Also: Appendix C, "Data Type and DDL Support on a Logical Standby Database"

- **Support for additional PL/SQL Package**
  - `DBMS_RLS` (row level security or Virtual Private Database)
  - `DBMS_FGA`

  See Also: Appendix C, "Data Type and DDL Support on a Logical Standby Database"

- **Support Transparent Data Encryption (TDE)**

  Data Guard SQL Apply can be used to provide data protection for the primary database with Transparent Data Encryption enabled. This allows a logical standby database to provide data protection for applications with advanced security requirements.

  See Also:

  - Chapter 10, "Managing a Logical Standby Database"
  - Section C.2, "Support for Transparent Data Encryption (TDE)"

- **Dynamic setting of Data Guard SQL Apply parameters**

  You can now configure specific SQL Apply parameters without requiring SQL Apply to be restarted. Using the `DBMS_LOGSTDBY.APPLY_SET` package, you can dynamically set initialization parameters, thus improving the manageability, uptime, and automation of a logical standby configuration.

  In addition, the `APPLY_SET` and `APPLY_UNSET` subprograms include two new parameters: `LOG_AUTO_DEL_RETENTION_TARGET` and `EVENT_LOG_DEST`.

  See Also: `DBMS_LOGSTDBY` PL/SQL package in the *Oracle Database PL/SQL Packages and Types Reference*

- **Enhanced RAC switchover support for logical standby databases**

  When switching over to a logical standby database where either the primary database or the standby database is using Oracle RAC, the `SWITCHOVER` command can be used without having to shut down any instance, either at the primary or at the logical standby database.

- **Enhanced DDL handling in Oracle Data Guard SQL Apply**

  SQL Apply will execute parallel DDLs in parallel (based on availability of parallel servers).

- **Use of the PL/SQL DBMS_SCHEDULER package to create Scheduler jobs on a standby database**

  Scheduler Jobs can be created on a standby database using the PL/SQL `DBMS_SCHEDULER` package and can be associated with an appropriate database role so that they run when intended (for example, when the database is the primary, standby, or both).

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 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:

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 (RAC) 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 RAC database.

The types of standby databases are as follows:

- **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, through **Redo Apply**, which recovers the redo data received from the primary database and applies the redo to the physical standby database.

  As of Oracle Database 11*g* release 1 (11.1), a physical standby database can receive and apply redo while it is open for read-only access. A physical standby database can therefore be used concurrently for data protection and reporting.

- **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 through **SQL Apply**, which transforms the data in the redo received from the primary database into SQL statements and then executes 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.

- **Snapshot Standby Database**

  A snapshot standby database is a fully updatable standby database that is created by converting a physical standby database into a snapshot standby database.

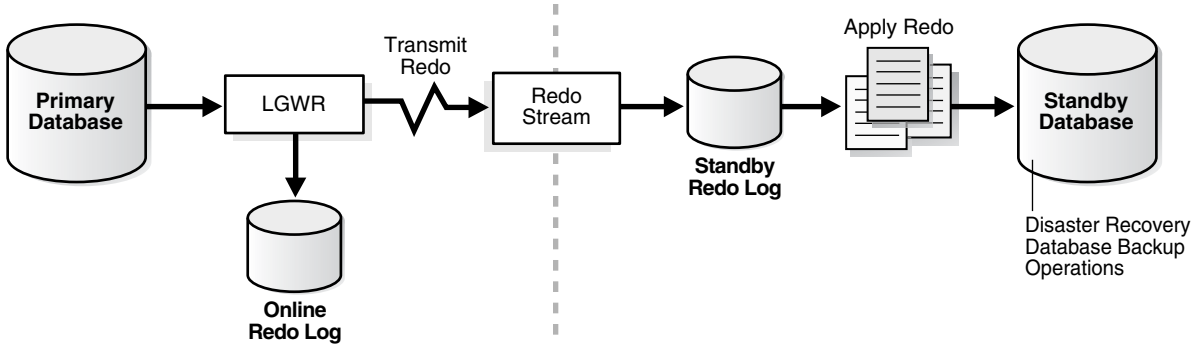
  Like a physical or logical standby database, a snapshot standby database receives and archives redo data from a primary database. Unlike a physical or logical standby database, a snapshot standby database does not apply the redo data that it receives. The redo data received by a snapshot standby database is not applied until the snapshot standby is converted back into a physical standby database, after first discarding any local updates made to the snapshot standby database.

A snapshot standby database is best used in scenarios that require a temporary, updatable snapshot of a physical standby database. Note that because redo data received by a snapshot standby database is not applied until it is converted back into a physical standby, the time needed to recover from a primary database failure is directly proportional to the amount of redo data that needs to be applied.

1.1.3 Configuration Example

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

Figure 1–1 Typical Data Guard Configuration



1.2 Data Guard Services

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

- Redo Transport Services

  Control the automated transfer of redo data from the production database to one or more archival destinations.

- Apply Services

  Apply redo data on the standby database to maintain transactional synchronization with the primary database. Redo data can be applied either from archived redo log files, or, if real-time apply is enabled, directly from the standby redo log files as they are being filled, without requiring the redo data to be archived first at the standby database.

- Role Transitions

  Change the role of a database from a standby database to a primary database, or from a primary database to a standby database using either a switchover or a failover operation.

1.2.1 Redo Transport Services

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

Redo transport services perform the following tasks:

- Transmit redo data from the primary system to the standby systems in the configuration
- Manage the process of resolving any gaps in the archived redo log files due to a network failure
- 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

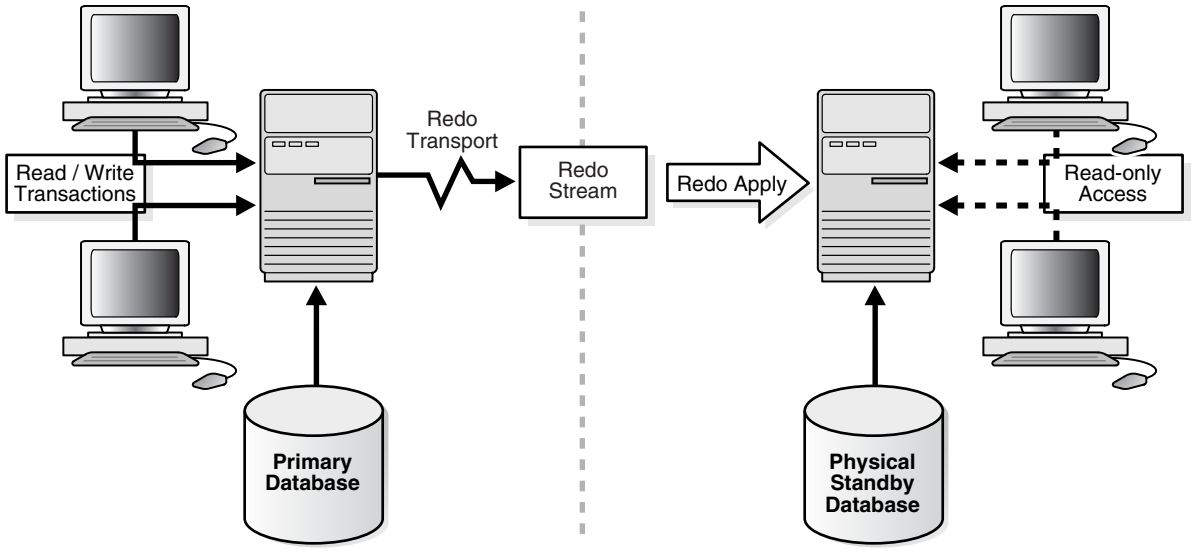
1.2.2 Apply Services

The redo data transmitted from the primary database is written to the standby redo log on the standby database. **Apply services** automatically apply the redo data on the standby database to maintain consistency with the primary database. It also allows read-only access to the data.

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

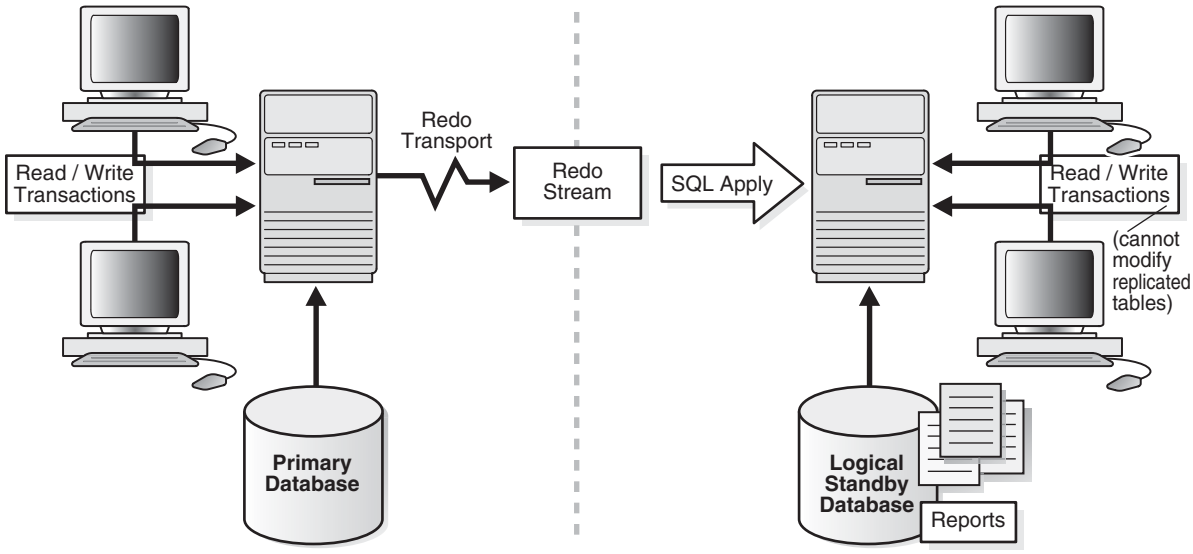
- For physical standby databases, Data Guard uses **Redo Apply** technology, which applies redo data on the standby database using standard recovery techniques of an Oracle database, as shown in Figure 1–2.

Figure 1–2 Automatic Updating of a Physical Standby Database



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

Figure 1–3 Automatic Updating of a Logical Standby Database



1.2.3 Role Transitions

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

A **switchover** is a role reversal between the primary database and one of its standby databases. A switchover ensures no data loss. This is typically done for planned maintenance of the primary system. During a switchover, the primary database transitions to a standby role, and the standby database transitions to the primary role.

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

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

1.3 Data Guard Broker

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

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

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

In addition, Oracle Enterprise Manager automates and simplifies:

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

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

1.3.1 Using Oracle Enterprise Manager Grid Control

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

From the Central Console of Enterprise Manager Grid Control, you can perform all management operations either 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 Also: *Oracle Data Guard Broker* for more information about the broker interface in Oracle Enterprise Manager Grid Control

1.3.2 Using the Data Guard Command-Line Interface

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

1.4 Data Guard Protection Modes

In some situations, a business cannot afford to lose data regardless of the circumstances. In other situations, the availability of the database may be more important than any potential data loss in the unlikely event of a multiple failure. Finally, some applications require maximum database performance at all times, and can therefore tolerate a small amount of data loss if any component should fail. The following descriptions summarize the three distinct modes of data protection.

Maximum availability This protection mode provides the highest level of data protection that is possible without compromising the availability of a primary database. Transactions do not commit until all redo data needed to recover those transactions has been written to the online redo log and to at least one standby database. If the primary database cannot write its redo stream to at least one synchronized standby database, it operates as if it were in maximum performance mode to preserve primary database availability until it is again able to write its redo stream to a synchronized standby database.

This protection mode ensures zero data loss except in the case of certain double faults, such as failure of a primary database after failure of the standby database.

Maximum performance This is the default protection mode. It provides the highest level of data protection that is possible without affecting the performance of a primary database. This is accomplished by allowing transactions to commit as soon as all redo data generated by those transactions has been written to the online log. Redo data is also written to one or more standby databases, but this is done asynchronously with respect to transaction commitment, so primary database performance is unaffected by delays in writing redo data to the standby database(s).

This protection mode offers slightly less data protection than maximum availability mode and has minimal impact on primary database performance.

Maximum protection This protection mode ensures that no data loss will occur if the primary database fails. To provide this level of protection, the redo data needed to recover a transaction must be written to both the online redo log and to at least one standby database before the transaction commits. To ensure that data loss cannot occur, the primary database will shut down, rather than continue processing transactions, if it cannot write its redo stream to at least one standby database.

All three protection modes require that specific redo transport options be used to send redo data to at least one standby database. See Chapter 5, "Data Guard Protection Modes" for more information about setting the protection mode of a primary database.

1.5 Client Failover

A high availability architecture requires a fast failover capability for databases and database clients.

Client failover encompasses failure notification, stale connection cleanup, and transparent reconnection to the new primary database. Oracle Database provides the capability to integrate database failover with failover procedures that automatically redirect clients to a new primary database within seconds of a database failover.

For more information about client failover, please refer to the Maximum Availability Architecture client failover best practices white paper at

`http://www.oracle.com/technology/deploy/availability/htdocs/maa.htm`

1.6 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)

Oracle 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. Oracle 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:

- Oracle 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 Oracle RAC and Data Guard are possible depending on the use of local and remote sites and the use of nodes and a combination of logical and physical standby databases. See Appendix D, "Data Guard and Oracle Real Application Clusters" and *Oracle Database High Availability Overview* for Oracle 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 User's Guide* for information about Flashback Database, and Section 7.2.2 for information describing the application of redo data.

- Recovery Manager (RMAN)

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

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

See Appendix F, "Creating a Standby Database with Recovery Manager" and *Oracle Database Backup and Recovery User's Guide*.

1.7 Summary of Data Guard Benefits

Data Guard offers these benefits:

- Disaster recovery, data protection, and high availability

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

- Complete data protection

  Data Guard can ensure zero data loss, even in the face of unforeseen disasters. A standby database provides a safeguard against data corruption and user errors. Because the redo data received from a primary database is validated at a standby database, 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.

- 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.

- Flexibility in data protection to balance availability against performance requirements

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

- Automatic gap detection and resolution

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

- Centralized and simple management

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

- Integration with Oracle Database

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

- Automatic role transitions

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

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

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 these types: a physical standby database, a logical standby database, or a snapshot standby database. If needed, either a physical or a logical standby database can assume the role of the primary database and take over production processing. A Data Guard configuration can include any combination of these types of standby databases.

2.1.1 Physical Standby Databases

A physical standby database is an exact, block-for-block copy of a primary database. A physical standby is maintained as an exact copy through a process called Redo Apply, in which redo data received from a primary database is continuously applied to a physical standby database using the database recovery mechanisms.

Benefits of a Physical Standby Database

A physical standby database provides the following benefits:

- Disaster recovery and high availability

  A physical standby database is 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

A physical standby database can prevent data loss, even in the face of unforeseen disasters. A physical standby database supports all datatypes, and all DDL and DML operations that the primary database can support. It also provides a safeguard against data corruptions and user errors. Storage level physical corruptions on the primary database will not be propagated to a standby database. Similarly, logical corruptions or user errors that would otherwise cause data loss can be easily resolved.

- Reduction in primary database workload

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

  A physical standby database can also be queried while Redo Apply is active, which allows queries to be offloaded from the primary to a physical standby, further reducing the primary workload.

- Performance

  The Redo Apply technology used by a physical standby database is the most efficient mechanism for keeping a standby database updated with changes being made at a primary database because it applies changes using low-level recovery mechanisms which bypass all SQL level code layers.

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. See Appendix C for information on data type and DDL support on logical standby databases.

Benefits of a Logical Standby Database

A logical standby database is ideal for high availability (HA) while still offering data recovery (DR) benefits. Compared to a physical standby database, a logical standby database provides significant additional HA benefits:

- Protection against additional kinds of failure

  Because logical standby analyzes the redo and reconstructs logical changes to the database, it can detect and protect against certain kinds of hardware failure on the primary that could potentially be replicated through block level changes. Oracle supports having both physical and logical standbys for the same primary server.

- Efficient use of resources

A logical standby database is open read/write while changes on the primary are being replicated. Consequently, a logical standby database can simultaneously be used to meet many other business requirements, for example it can run reporting workloads that would problematical for the primary's throughput. It can be used to test new software releases and some kinds of applications on a complete and accurate copy of the primary's data. It can host other applications and additional schemas while protecting data replicated from the primary against local changes. It can be used to assess the impact of certain kinds of physical restructuring (for example, changes to partitioning schemes). Because a logical standby identifies user transactions and replicates only those changes while filtering out background system changes, it can efficiently replicate only transactions of interest.

- Workload distribution

  Logical standby provides a simple turnkey solution for creating up-to-the-minute, consistent replicas of a primary database that can be used for workload distribution. As the reporting workload increases, additional logical standbys can be created with transparent load distribution without affecting the transactional throughput of the primary server.

- Optimized for reporting and decision support requirements

  A key benefit of logical standby is that significant auxiliary structures can be created to optimize the reporting workload; structures that could have a prohibitive impact on the primary's transactional response time. A logical standby can have its data physically reorganized into a different storage type with different partitioning, have many different indexes, have on-demand refresh materialized views created and maintained, and it can be used to drive the creation of data cubes and other OLAP data views.

- Minimizing downtime on software upgrades

  Logical standby can be used to greatly reduce downtime associated with applying patchsets and new software releases. A logical standby can be upgraded to the new release and then switched over to become the active primary. This allows full availability while the old primary is converted to a logical standby and the patchset is applied.

2.1.3 Snapshot Standby Databases

A snapshot standby database is a fully updatable standby database that is created by converting a physical standby database into a snapshot standby database. A snapshot standby database receives and archives, but does not apply, redo data from its primary database. Redo data received from the primary database is applied when a snapshot standby database is converted back into a physical standby database, after discarding all local updates to the snapshot standby database.

A snapshot standby database typically diverges from its primary database over time because redo data from the primary database is not applied as it is received. Local updates to the snapshot standby database will cause additional divergence. The data in the primary database is fully protected however, because a snapshot standby can be converted back into a physical standby database at any time, and the redo data received from the primary will then be applied.

Benefits of a Snapshot Standby Database

A snapshot standby database is a fully updatable standby database that provides disaster recovery and data protection benefits that are similar to those of a physical standby database. Snapshot standby databases are best used in scenarios where the

benefit of having a temporary, updatable snapshot of the primary database justifies additional administrative complexity and increased time to recover from primary database failures.

The benefits of using a snapshot standby database include the following:

- It provides an exact replica of a production database for development and testing purposes, while maintaining data protection at all times.
- It can be easily refreshed to contain current production data by converting to a physical standby and resynchronizing.

The ability to create a snapshot standby, test, resynchronize with production, and then again create a snapshot standby and test, is a cycle that can be repeated as often as desired. The same process can be used to easily create and regularly update a snapshot standby for reporting purposes where read/write access to data is required.

2.2 User Interfaces for Administering Data Guard Configurations

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

- Oracle Enterprise Manager

  Enterprise Manager provides a GUI interface for the Data Guard broker that automates many of the tasks involved in creating, configuring, and monitoring a Data Guard environment. See *Oracle Data Guard Broker* and the Oracle Enterprise Manager online Help for information about the GUI and its wizards.

- SQL*Plus Command-line interface

  Several SQL*Plus statements use the `STANDBY` keyword to specify operations on a standby database. Other SQL statements do not include standby-specific syntax, but they are useful for performing operations on a standby database. See Chapter 16 for a list of the relevant statements.

- Initialization parameters

  Several initialization parameters are used to define the Data Guard environment. See Chapter 14 for a list of the relevant initialization parameters.

- Data Guard broker command-line interface (DGMGRL)

  The DGMGRL command-line interface is an alternative to using Oracle Enterprise Manager. The DGMGRL command-line interface is useful if you want to use the broker to manage a Data Guard configuration from batch programs or scripts. See *Oracle Data Guard Broker* for complete information.

2.3 Data Guard Operational Prerequisites

The following sections describe operational requirements for using Data Guard:

- Hardware and Operating System Requirements
- Oracle Software Requirements

2.3.1 Hardware and Operating System Requirements

As of Oracle Database 11*g*, and subject to current restrictions documented in MetaLink note 4134841.1, Data Guard provides increased flexibility for Data Guard configurations in which the primary and standby systems may have different CPU architectures, operating systems (for example, Windows & Linux), operating system

binaries (32-bit/64-bit), or Oracle database binaries (32-bit/64-bit). For specific information about mixed-platform support, see Oracle MetaLink note 413484.1 at https://metalink.oracle.com.

The same release of Oracle Database Enterprise Edition must be installed on the primary database and all standby databases, except during rolling database upgrades using logical standby databases.

See Also:

- Chapter 12, "Using SQL Apply to Upgrade the Oracle Database" for information about rolling database upgrades

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.

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

- Using Data Guard SQL Apply, you will be able to perform a rolling upgrade of the Oracle database software from patch set release *n* (minimally, this must be release 10.1.0.3) to any higher versioned patch set or major version release. During a rolling upgrade, you can run different releases of the Oracle database on the primary and logical standby databases while you upgrade them, one at a time. For complete information, see Chapter 12, "Using SQL Apply to Upgrade the Oracle Database" and the ReadMe file for the applicable Oracle Database 10*g* patch set release.
- The `COMPATIBLE` initialization parameter must be set to the same value on all databases in a Data Guard configuration.
- If you are currently running Oracle Data Guard on Oracle8*i* database software, see *Oracle Database Upgrade Guide* for complete information about upgrading to Oracle Data Guard 11*g*.
- The primary database must run in ARCHIVELOG mode. See *Oracle Database Administrator's Guide* for more information.
- The primary database can be a single instance database or an Oracle Real Application Cluster (RAC) database. The standby databases can be single instance databases or Oracle RAC databases, and these standby databases can be a mix of physical, logical, and snapshot types. See *Oracle Database High Availability Overview* for more information about configuring and using Oracle Data Guard with RAC.
- Each primary database and standby database must have its own control file.
- If a standby database is located on the same system as the primary database, the archival directories for the standby database *must* use a different directory

structure than the primary database. Otherwise, the standby database may overwrite the primary database files.

- To protect against unlogged direct writes in the primary database that cannot be propagated to the standby database, turn on `FORCE LOGGING` at the primary database before performing datafile backups for standby creation. Keep the database in `FORCE LOGGING` mode as long as the standby database is required.
- The user accounts you use to manage the primary and standby database instances must have `SYSDBA` system privileges.
- Oracle recommends that when you set up Oracle Automatic Storage Management (ASM) and Oracle Managed Files (OMF) in a Data Guard configuration, set it up symmetrically on the primary and standby database. That is, if any database in the Data Guard configuration uses ASM, OMF, or both, then every database in the configuration should use ASM, OMF, or both, respectively. See the scenario in Section 13.5 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.

> **See Also:** Your operating system-specific Oracle documentation for more information about Optimal Flexible Architecture (OFA)

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 13 for a scenario describing how to set up OMF in a Data Guard configuration.

Figure 2–1 Possible Standby Configurations

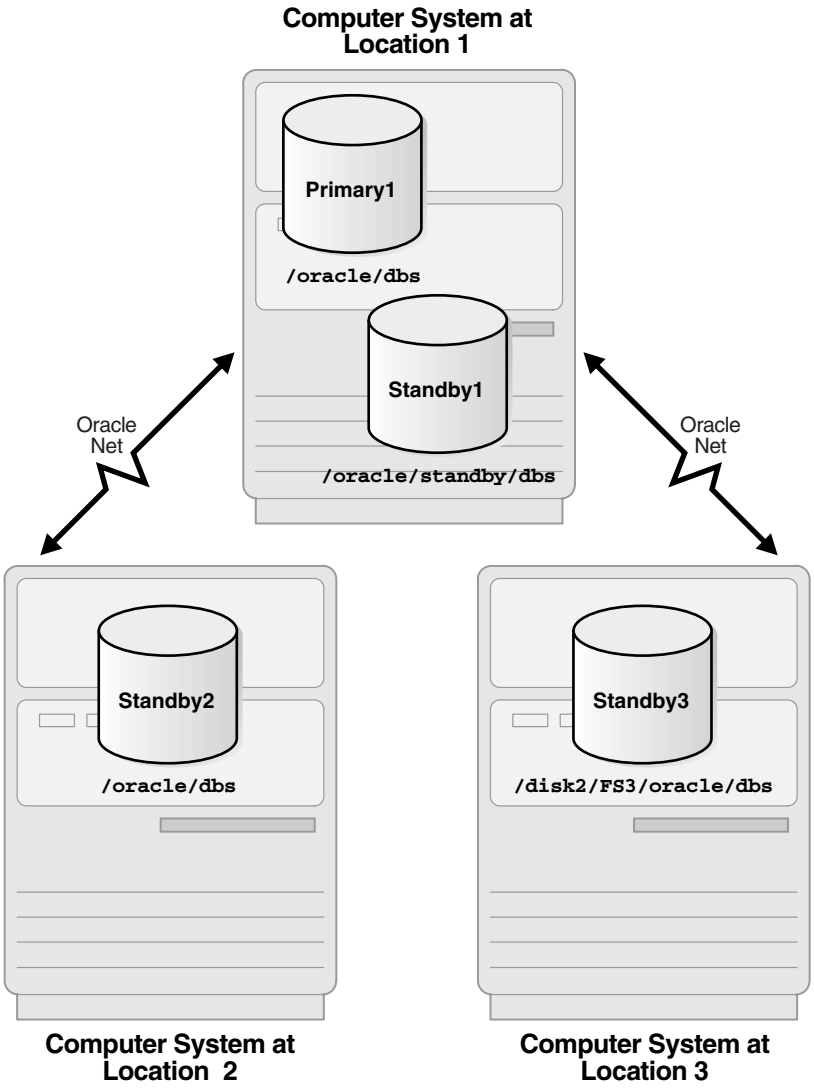


Table 2–1 describes possible configurations of primary and standby databases and the consequences of each. You must specify a unique value for the `DB_UNIQUE_NAME` initialization parameter when more than one member of a Data Guard configuration resides on the same system. Oracle recommends that the value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

Table 2–1 Standby Database Location and Directory Options

Standby System	Directory Structure	Consequences
Same as primary system	Different than primary system (required)	■ 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 3.1.4.) ■ The standby database does not protect against disasters that destroy the system on which the primary and standby databases reside, but it does provide switchover capabilities for planned maintenance.
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 3.1.4). ■ By locating the standby database on separate physical media, you safeguard the data on the primary database against disasters that destroy the primary system.

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
- Step-by-Step Instructions for Creating a Physical Standby Database
- Post-Creation Steps

The steps described in this chapter configure the standby database for maximum performance mode, which is the default data protection mode. Chapter 5 provides information about configuring the different data protection modes.

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
- Appendix F for information about creating a standby database with Recovery Manager (RMAN)

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	Configure Redo Transport Authentication
Section 3.1.3	Configure the Primary Database to Receive Redo Data
Section 3.1.4	Set Primary Database Initialization Parameters
Section 3.1.5	Enable Archiving

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

3.1.1 Enable Forced Logging

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

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

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

3.1.2 Configure Redo Transport Authentication

Data Guard uses Oracle Net sessions to transport redo data and control messages between the members of a Data Guard configuration. These redo transport sessions are authenticated using either the Secure Sockets Layer (SSL) protocol or a remote login password file.

SSL is used to authenticate redo transport sessions between two databases if:

- The databases are members of the same Oracle Internet Directory (OID) enterprise domain and it allows the use of current user database links
- The `LOG_ARCHIVE_DEST_`*n*, `FAL_SERVER`, and `FAL_CLIENT` database initialization parameters that correspond to the databases use Oracle Net connect descriptors configured for SSL
- Each database has an Oracle wallet or supported hardware security module that contains a user certificate with a distinguished name (DN) that matches the DN in the OID entry for the database

If the SSL authentication requirements are not met, each member of a Data Guard configuration must be configured to use a remote login password file and every physical standby database in the configuration must have an up-to-date copy of the password file from the primary database.

Note that whenever you grant or revoke the `SYSDBA` or `SYSOPER` privilege or change the login password of a user who has these privileges, you must replace the password file at each physical or snapshot standby database in the configuration with a fresh copy of the password file from the primary database.

See Also:

- *Oracle Database Administrator's Guide* for more information about remote login password files
- *Oracle Database Advanced Security Administrator's Guide*
- *Oracle Database Net Services Administrator's Guide*

3.1.3 Configure the Primary Database to Receive Redo Data

Although this task is optional, Oracle recommends that a primary database be configured to receive redo data when a Data Guard configuration is created. By following this best practice, your primary database will be ready to quickly transition to the standby role and begin receiving redo data.

See Section 6.2.3 for a complete discussion of how to configure a database to receive redo data.

3.1.4 Set Primary Database Initialization Parameters

On the primary database, you define initialization parameters that control redo transport services while the database is in the primary role. There are additional parameters you need to add that control the receipt of the redo data and 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
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 ASYNC
  VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)
  DB_UNIQUE_NAME=boston'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_STATE_2=ENABLE
REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE
LOG_ARCHIVE_FORMAT=%t_%s_%r.arc
LOG_ARCHIVE_MAX_PROCESSES=30
```

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

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='boston','chicago'
LOG_FILE_NAME_CONVERT=
```

```
 '/arch1/boston/','/arch1/chicago/','/arch2/boston/','/arch2/chicago/'
STANDBY_FILE_MANAGEMENT=AUTO
```

Specifying the initialization parameters shown in Example 3–2 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 Example 3–1 and Example 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.
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 an Oracle RAC 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.
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.
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. Also, see Chapter 15 for complete LOG_ARCHIVE_DEST_*n* information.
LOG_ARCHIVE_DEST_STATE_n	Specify ENABLE to allow redo transport services to transmit redo data to the specified destination.
REMOTE_LOGIN_PASSWORDFILE	This parameter must be set to EXCLUSIVE or SHARED if a remote login password file is used to authenticate administrative users or redo transport sessions.
LOG_ARCHIVE_FORMAT	Specify the format for the archived redo log files using a thread (%t), sequence number (%s), and resetlogs ID (%r).
LOG_ARCHIVE_MAX_PROCESSES =integer	Specify the maximum number (from 1 to 30) of archiver (ARC*n*) processes you want Oracle software to invoke initially. The default value is 4.
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.

Parameter	Recommended Setting
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.
DB_FILE_NAME_CONVERT	Specify the path name and filename location of the primary database datafiles followed by the standby location. This parameter converts the path names of the primary database datafiles to the standby datafile path names. If the standby database is on the same system as the primary database or if the directory structure where the datafiles are located on the standby site is different from the primary site, then this parameter is required. Note that this parameter is used only to convert path names for physical standby databases. Multiple pairs of paths may be specified by this parameter.
LOG_FILE_NAME_CONVERT	Specify the location of the primary database online redo log files followed by the standby location. This parameter converts the path names of the primary database log files to the path names on the standby database. If the standby database is on the same system as the primary database or if the directory structure where the log files are located on the standby system is different from the primary system, then this parameter is required. Multiple pairs of paths may be specified by this parameter.
STANDBY_FILE_MANAGEMENT	Set to AUTO so when datafiles are added to or dropped from the primary database, corresponding changes are made automatically to the standby database.

Caution: Review the initialization parameter file for additional parameters that may need to be modified. For example, you may need to modify the dump destination parameters if the directory location on the standby database is different from those specified on the primary database.

3.1.5 Enable Archiving

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

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

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

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

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

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

Table 3–2 Creating a Physical Standby Database

Reference	Task	Database
Section 3.2.1	Create a Backup Copy of the Primary Database Datafiles	Primary
Section 3.2.2	Create a Control File for the Standby Database	Primary
Section 3.2.3	Prepare an Initialization Parameter File for the Standby Database	Primary

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 Database High Availability Architecture and Best Practices* for backup recommendations and *Oracle Database Backup and Recovery User's Guide* to perform a database 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
LOG_ARCHIVE_CONFIG='DG_CONFIG=(chicago,boston)'
CONTROL_FILES='/arch1/boston/control1.ctl', '/arch2/boston/control2.ctl'
DB_FILE_NAME_CONVERT='chicago','boston'
LOG_FILE_NAME_CONVERT=
 '/arch1/chicago/','/arch1/boston/','/arch2/chicago/','/arch2/boston/'
LOG_ARCHIVE_FORMAT=log%t_%s_%r.arc
LOG_ARCHIVE_DEST_1=
 'LOCATION=/arch1/boston/
  VALID_FOR=(ALL_LOGFILES,ALL_ROLES)
  DB_UNIQUE_NAME=boston'
LOG_ARCHIVE_DEST_2=
 'SERVICE=chicago ASYNC
  VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)
  DB_UNIQUE_NAME=chicago'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_STATE_2=ENABLE
REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE
STANDBY_FILE_MANAGEMENT=AUTO
FAL_SERVER=chicago
FAL_CLIENT=boston
.
.
.
```

Ensure the COMPATIBLE initialization parameter is set to the same value on both the primary and standby databases. If the values differ, redo transport services may be unable to transmit redo data from the primary database to the standby databases. In a Data Guard configuration, COMPATIBLE must be set to a minimum of 9.2.0.1.0. However, if you want to take advantage of new Oracle Database 11*g* features, set the COMPATIBLE parameter to 11.0.0.

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.
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. Also, see Chapter 15 for complete information about LOG_ARCHIVE_DEST_*n*.
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.
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.

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 if the directory location on the standby database is different from those specified on the primary database.

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 database will be hosted on a Windows system, use the ORADIM utility to create a Windows service. For example:

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

See *Oracle Database Platform Guide for Microsoft Windows* for more information about using the ORADIM utility.

Step 2 Copy the remote login password file from the primary database system to the standby database system

If the primary database has a remote login password file, copy it to the appropriate directory on the physical standby database system. Note that the password file must be re-copied each time the `SYSDBA` or `SYSOPER` privilege is granted or revoked and whenever the login password of a user with these privileges is changed.

This step is optional if operating system authentication is used for administrative users and if SSL is used for redo transport authentication

Step 3 Configure listeners for the primary and standby databases.

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

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

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

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

Step 4 Create Oracle Net service names.

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

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

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

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

On an idle standby database, use the SQL `CREATE` statement to create a server parameter file for the standby database from the text initialization parameter file that was edited in Step 2 on page 3-6. For example:

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

Step 6 Copy the primary database encryption wallet to the standby database system

If the primary database has a database encryption wallet, copy it to the standby database system and configure the standby database to use this wallet.

Note: The database encryption wallet must be copied from the primary database system to each standby database system whenever the master encryption key is updated.

Encrypted data in a standby database cannot be accessed unless the standby database is configured to point to a database encryption wallet or hardware security module that contains the current master encryption key from the primary database.

See Also: *Oracle Database Advanced Security Administrator's Guide* for more information about transparent database encryption

3.2.6 Start the Physical Standby Database

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

Step 1 Start the physical standby database.

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

```
SQL> STARTUP MOUNT;
```

Step 2 Prepare the Standby Database to Receive Redo Data

Prepare the standby database to receive and archive redo data from the primary database, by performing the steps described in Section 6.2.3.

Step 3 Create an Online Redo Log on the Standby Database

Although this step is optional, Oracle recommends that an online redo log be created when a standby database is created. By following this best practice, a standby database will be ready to quickly transition to the primary database role.

The size and number of redo log groups in the online redo log of a standby database should be chosen so that the standby database performs well if it transitions to the primary role.

Step 4 Start Redo Apply.

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

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

The statement includes the `DISCONNECT FROM SESSION` option so that Redo Apply runs in a background session. See Section 7.3, "Applying Redo Data to Physical Standby Databases" for more information.

The statement also includes the `USING CURRENT LOGFILE` clause so that redo can be applied as soon as it has been received. See Section 7.3.1, "Starting Redo Apply" for more information.

3.2.7 Verify the Physical Standby Database Is Performing Properly

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

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

Step 1 Identify the existing archived redo log files.

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

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

 SEQUENCE# FIRST_TIME         NEXT_TIME
---------- ------------------ ------------------
         8 11-JUL-07 17:50:45 11-JUL-07 17:50:53
         9 11-JUL-07 17:50:53 11-JUL-07 17:50:58
        10 11-JUL-07 17:50:58 11-JUL-07 17:51:03

3 rows selected.
```

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

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

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

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

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

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

 SEQUENCE# FIRST_TIME         NEXT_TIME
---------- ------------------ ------------------
         8 11-JUL-07 17:50:45 11-JUL-07 17:50:53
         9 11-JUL-07 17:50:53 11-JUL-07 17:50:58
        10 11-JUL-07 17:50:58 11-JUL-07 17:51:03
        11 11-JUL-07 17:51:03 11-JUL-07 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 that received redo has been applied.

On the standby database, query the V$ARCHIVED_LOG view to verify that received redo has been applied:

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

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

4 rows selected.

Note: The value of the `APPLIED` column for the most recently received log file will be either `IN-MEMORY` or `YES` if that log file has been applied.

3.3 Post-Creation Steps

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

- Upgrade the data protection mode

  The Data Guard configuration is initially set up in the maximum performance mode (the default).

- Enable Flashback Database

  Flashback Database removes the need to re-create the primary database after a failover. Flashback Database enables you to return a database to its state at a time in the recent past much faster than traditional point-in-time recovery, because it does not require restoring datafiles from backup nor the extensive application of redo data. You can enable Flashback Database on the primary database, the standby database, or both. See Section 13.2 and Section 13.3 for scenarios showing how to use Flashback Database in a Data Guard environment. Also, see *Oracle Database Backup and Recovery User's Guide* for more information about Flashback Database.

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:

- Prerequisite Conditions for Creating a Logical Standby Database
- Step-by-Step Instructions for Creating a Logical Standby Database
- Post-Creation Steps

See Also:

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

4.1 Prerequisite Conditions for Creating a Logical Standby Database

Before you create a logical standby database, you must first ensure the primary database is properly configured. Table 4–1 provides a checklist of the tasks that you perform on the primary database to prepare for logical standby database creation.

Table 4–1 Preparing the Primary Database for Logical Standby Database Creation

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

Note that a logical standby database uses standby redo logs (SRLs) for redo received from the primary database, and also writes to online redo logs (ORLs) as it applies changes to the standby database. Thus, logical standby databases often require additional ARC*n* processes to simultaneously archive SRLs and ORLs. Additionally, because archiving of ORLs takes precedence over archiving of SRLs, a greater number of SRLs may be needed on a logical standby during periods of very high workload.

4.1.1 Determine Support for Data Types and Storage Attributes for Tables

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

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

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

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

- If a table has a primary key defined, then the primary key is logged along with the modified columns as part of the `UPDATE` statement to identify the modified row.
- In the absence of a primary key, the shortest nonnull unique-constraint/index is logged along with the modified columns as part of the `UPDATE` statement to identify the modified row.
- In the absence of both a primary key and a nonnull unique constraint/index, all columns of bounded size are logged as part of the `UPDATE` statement to identify the modified row. In other words, all columns except those with the following types are logged: `LONG`, `LOB`, `LONG RAW`, object type, and collections.
- A function-based index, even though it is declared as unique, cannot be used to uniquely identify a modified row. However, logical standby databases support replication of tables that have function-based indexes defined, as long as modified rows can be uniquely identified.

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

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

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

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

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

Step 2 Add a disabled primary-key RELY constraint.

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

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

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

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

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

See Also:

- *Oracle Database Reference* for information about the DBA_LOGSTDBY_NOT_UNIQUE view
- *Oracle Database SQL Language Reference* for information about the ALTER TABLE statement syntax and creating RELY constraints
- Section 10.7.1, "Create a Primary Key RELY Constraint" on page 10-26 for information about RELY constraints and actions you can take to increase performance on a logical standby database

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

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

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

Table 4–2 Creating a Logical Standby Database

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

4.2.1 Create a Physical Standby Database

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

4.2.2 Stop Redo Apply on the Physical Standby Database

You can run Redo Apply on the new physical standby database for any length of time before converting it to a logical standby database. However, before converting to a logical standby database, stop Redo Apply on the physical standby database. Stopping Redo Apply is necessary to avoid applying changes past the redo that contains the LogMiner dictionary (described in Section 4.2.3.2, "Build a Dictionary in the Redo Data" on page 4-5).

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

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

4.2.3 Prepare the Primary Database to Support a Logical Standby Database

This section contains the following topics:

- Prepare the Primary Database for Role Transitions
- Build a Dictionary in the Redo Data

4.2.3.1 Prepare the Primary Database for Role Transitions

In Section 3.1.4, "Set Primary Database Initialization Parameters" on page 3-3, you set up several standby role initialization parameters to take effect when the primary database is transitioned to the *physical* standby role.

> **Note:** This step is necessary only if you plan to perform switchovers.

If you plan to transition the primary database to the *logical* standby role, then you must also modify the parameters shown in bold typeface in Example 4–1, so that no parameters need to change after a role transition:

- Change the `VALID_FOR` attribute in the original `LOG_ARCHIVE_DEST_1` destination to archive redo data only from the online redo log and not from the standby redo log.
- Include the `LOG_ARCHIVE_DEST_3` destination on the primary database. This parameter only takes effect when the primary database is transitioned to the logical standby role.

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_3=
 'LOCATION=/arch2/chicago/
  VALID_FOR=(STANDBY_LOGFILES,STANDBY_ROLE)
  DB_UNIQUE_NAME=chicago'
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

To dynamically set these initialization parameter, use the SQL `ALTER SYSTEM SET` statement and include the `SCOPE=BOTH` clause so that the changes take effect immediately and persist after the database is shut down and started up again.

The following table describes the archival processing defined by the changed 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	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/chicago/.
LOG_ARCHIVE_DEST_3	Is ignored; LOG_ARCHIVE_DEST_3 is valid only when chicago is running in the standby role.	Directs archiving of redo data from the standby redo log files to the local archived redo log files in /arch2/chicago/.

4.2.3.2 Build a Dictionary in the Redo Data

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

To build the LogMiner dictionary, issue the following statement:

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

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

See Also:

- The DBMS_LOGSTDBY.BUILD PL/SQL package in *Oracle Database PL/SQL Packages and Types Reference*
- The UNDO_RETENTION initialization parameter in *Oracle Database Reference*

4.2.4 Transition to a Logical Standby Database

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

- Convert to a Logical Standby Database
- Adjust Initialization Parameters for the Logical Standby Database

4.2.4.1 Convert to a Logical Standby Database

The redo logs contain the information necessary to convert your physical standby database to a logical standby database.

Note: If you have an Oracle RAC physical standby database, shut down all but one instance, set CLUSTER_DATABASE to FALSE, and start the standby database as a single instance in MOUNT EXCLUSIVE mode, as follows:

```
SQL> ALTER SYSTEM SET CLUSTER_DATABASE=FALSE SCOPE=SPFILE;
SQL> SHUTDOWN ABORT;
SQL> STARTUP MOUNT EXCLUSIVE;
```

To continue applying redo data to the physical standby database until it is ready to convert to a logical standby database, issue the following SQL statement:

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

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

Note: If you are creating a logical standby database in the context of performing a rolling upgrade of Oracle software with a physical standby database, you should issue the following command instead:

```
SQL> ALTER DATABASE RECOVER TO LOGICAL STANDBY KEEP IDENTITY;
```

A logical standby database created with the KEEP IDENTITY clause retains the same DB_NAME and DBID as that of its primary database. Such a logical standby database can only participate in one switchover operation, and thus should only be created in the context of a rolling upgrade with a physical standby database.

Note that the KEEP IDENTITY clause is available only if the database being upgraded is running Oracle Database release 11.1 or later.

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

Caution: In earlier releases, you needed to create a new password file before you opened the logical standby database. This is no longer needed. Creating a new password file at the logical standby database will cause redo transport services to not work properly.

4.2.4.2 Adjust Initialization Parameters for the Logical Standby Database

---

Note: If you started with an Oracle RAC physical standby database, set CLUSTER_DATABASE back to TRUE, as follows:

```
SQL> ALTER SYSTEM SET CLUSTER_DATABASE=TRUE SCOPE=SPFILE;
```

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

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

You need to modify the LOG_ARCHIVE_DEST_*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 parameters 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 ASYNC
   VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)
   DB_UNIQUE_NAME=chicago'
LOG_ARCHIVE_DEST_3=
  'LOCATION=/arch2/boston/
   VALID_FOR=(STANDBY_LOGFILES,STANDBY_ROLE)
   DB_UNIQUE_NAME=boston'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_STATE_2=ENABLE
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

Note: If database compatibility is set to 11.1, you can also use the Flash Recovery Area to store the remote archived logs. To do this, set the following parameters (assuming you have already appropriately set DB_RECOVERY_FILE_DEST and DB_RECOVERY_FILE_DEST_SIZE):

```
LOG_ARCHIVE_DEST_1=
  'LOCATION=USE_DB_RECOVERY_FILE_DEST
   VALID_FOR=(ONLINE_LOGFILES, ALL_ROLES)
   DB_UNIQUE_NAME=boston'
LOG_ARCHIVE_DEST_3=
  'LOCATION=USE_DB_RECOVERY_FILE_DEST
   VALID_FOR=(STANDBY_LOGFILES, STANDBY_ROLE)
   DB_UNIQUE_NAME=boston'
```

The following table describes the archival processing defined by the initialization parameters shown in Example 4–2.

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

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

4.2.5 Open the Logical Standby Database

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

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

Note: If you started with a Oracle RAC physical standby database, you can start up all other standby instances at this point.

Caution: If you are co-locating the logical standby database on the same computer system as the primary database, you must issue the following SQL statement before starting SQL Apply for the first time, so that SQL Apply skips the file operations performed at the primary database. The reason this is necessary is that SQL Apply has access to the same directory structure as the primary database, and datafiles that belong to the primary database could possibly be damaged if SQL Apply attempted to reexecute certain file-specific operations.

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP('ALTER TABLESPACE');
```

The DB_FILENAME_CONVERT parameter that you set up while co-locating the physical standby database on the same system as the primary database, is ignored by SQL Apply. See *Oracle Database PL/SQL Packages and Types Reference* for information about DBMS_LOGSTDBY.SKIP and equivalent behavior in the context of a logical standby database.

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

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

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

4.2.6 Verify the Logical Standby Database Is Performing Properly

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

- Chapter 6, "Redo Transport Services"
- Chapter 10, "Managing a Logical Standby Database"

4.3 Post-Creation Steps

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

- Upgrade the data protection mode

  The Data Guard configuration is initially set up in the maximum performance mode (the default).

- Enable Flashback Database

  Flashback Database removes the need to re-create the primary database after a failover. Flashback Database enables you to return a database to its state at a time in the recent past much faster than traditional point-in-time recovery, because it does not require restoring datafiles from backup nor the extensive application of redo data. You can enable Flashback Database on the primary database, the standby database, or both. See Section 13.2, "Converting a Failed Primary Into a Standby Database Using Flashback Database" on page 13-4 and Section 13.3, "Using Flashback Database After Issuing an Open Resetlogs Statement" on page 13-8 for scenarios showing how to use Flashback Database in a Data Guard environment. Also, see *Oracle Database Backup and Recovery User's Guide* for more information about Flashback Database.

5
Data Guard Protection Modes

This chapter contains the following sections:

5.1 Data Guard Protection Modes

This section describes the Data Guard protection modes.

In these descriptions, a synchronized standby database is meant to be one that meets the minimum requirements of the configured data protection mode and that does not have a redo gap. Redo gaps are discussed in Section 6.3.3.

Maximum Availability

This protection mode provides the highest level of data protection that is possible without compromising the availability of a primary database. Transactions do not commit until all redo data needed to recover those transactions has been written to the online redo log and to at least one synchronized standby database. If the primary database cannot write its redo stream to at least one synchronized standby database, it operates as if it were in maximum performance mode to preserve primary database availability until it is again able to write its redo stream to a synchronized standby database.

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

Maximum Performance

This protection mode provides the highest level of data protection that is possible without affecting the performance of a primary database. This is accomplished by allowing transactions to commit as soon as all redo data generated by those transactions has been written to the online log. Redo data is also written to one or more standby databases, but this is done asynchronously with respect to transaction commitment, so primary database performance is unaffected by delays in writing redo data to the standby database(s).

This protection mode offers slightly less data protection than maximum availability mode and has minimal impact on primary database performance.

This is the default protection mode.

Maximum Protection

This protection mode ensures that zero data loss occurs if a primary database fails. To provide this level of protection, the redo data needed to recover a transaction must be written to both the online redo log and to at least one synchronized standby database before the transaction commits. To ensure that data loss cannot occur, the primary database will shut down, rather than continue processing transactions, if it cannot write its redo stream to at least one synchronized standby database.

Because this data protection mode prioritizes data protection over primary database availability, Oracle recommends that a minimum of two standby databases be used to protect a primary database that runs in maximum protection mode to prevent a single standby database failure from causing the primary database to shut down.

5.2 Setting the Data Protection Mode of a Primary Database

Perform the following steps to change the data protection mode of a primary database:

Step 1 Select a data protection mode that meets your availability, performance and data protection requirements.

See Section 5.1 for a description of the available modes.

Step 2 Verify that redo transport is configured to at least one standby database

The value of the LOG_ARCHIVE_DEST_*n* database initialization parameter that corresponds to the standby database must include the redo transport attributes listed in Table 5–1 for the data protection mode that you are moving to.

If the primary database has more than one standby database, only one of those standby databases must use the redo transport settings listed in Table 5–1.

The standby database must also have a standby redo log.

See Chapter 6, "Redo Transport Services" for more information about configuring redo transport and standby redo logs.

Table 5–1 Required Redo Transport Attributes for Data Protection Modes

Maximum Availability	Maximum Performance	Maximum Protection
AFFIRM	NOAFFIRM	AFFIRM
SYNC	ASYNC	SYNC
DB_UNIQUE_NAME	DB_UNIQUE_NAME	DB_UNIQUE_NAME

Step 3 Verify that the DB_UNIQUE_NAME database initialization parameter has been set to a unique name on the primary and standby database.

For example, if the DB_UNIQUE_NAME parameter has not been defined on either database, the following SQL statements might be used to assign a unique name to each database.

Execute this SQL statement on the primary database:

```
SQL> ALTER SYSTEM SET DB_UNIQUE_NAME='CHICAGO' SCOPE=SPFILE;
```

Execute this SQL statement on the standby database:

```
SQL> ALTER SYSTEM SET DB_UNIQUE_NAME='BOSTON' SCOPE=SPFILE;
```

Step 4 Verify that the LOG_ARCHIVE_CONFIG database initialization parameter has been defined on the primary and standby database and that its value includes a DG_CONFIG list that includes the DB_UNIQUE_NAME of the primary and standby database.

For example, if the LOG_ARCHIVE_CONFIG parameter has not been defined on either database, the following SQL statement could be executed on each database to configure the LOG_ARCHIVE_CONFIG parameter:

```
SQL> ALTER SYSTEM SET
  2> LOG_ARCHIVE_CONFIG='DG_CONFIG=(CHICAGO,BOSTON)';
```

Step 5 Shut down the primary database and restart it in mounted mode if the protection mode is being set to Maximum Protection or being changed from Maximum Performance to Maximum Availability. If the primary database is an Oracle Real Applications Cluster, shut down all of the instances and then start and mount a single instance.

For example:

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

Step 6 Set the data protection mode.

Execute the following SQL statement on the primary database:

```
SQL> ALTER DATABASE
  2> SET STANDBY DATABASE TO MAXIMIZE {AVAILABILITY | PERFORMANCE | PROTECTION};
```

If the primary database is an Oracle Real Applications Cluster, any instances stopped in Step 5 can now be restarted.

Step 7 Open the primary database.

If the database was restarted in Step 5, open the database:

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

Step 8 Confirm that the primary database is operating in the new protection mode.

Perform the following query on the primary database to confirm that it is operating in the new protection mode:

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

6

Redo Transport Services

This chapter describes how to configure and monitor Oracle redo transport services. The following topics are discussed:

6.1 Introduction to Redo Transport Services

Redo transport services performs the automated transfer of redo data between Oracle databases. The following redo transport destinations are supported:

- Oracle Data Guard standby databases

  This guide describes how to create and manage standby databases.

- Archive Log repository

  This destination type is used for temporary offsite storage of archived redo log files. An archive log repository consists of an Oracle database instance and a physical standby control file. An archive log repository does not contain datafiles, so it cannot support role transitions.

  The procedure used to create an archive log repository is identical to the procedure used to create a physical standby database, except for the copying of datafiles.

- Oracle Streams downstream capture databases

  See *Oracle Streams Concepts and Administration* for more information about Oracle Streams downstream capture databases.

- Oracle Change Data Capture staging databases

  See *Oracle Warehouse Builder User's Guide* for more information about Oracle Change Data Capture staging databases.

An Oracle database can send redo data to up to nine redo transport destinations. Each redo transport destination is individually configured to receive redo data via one of two redo transport modes:

- Synchronous

  The synchronous redo transport mode transmits redo data synchronously with respect to transaction commitment. A transaction cannot commit until all redo

generated by that transaction has been successfully sent to every enabled redo transport destination that uses the synchronous redo transport mode.

This transport mode is used by the Maximum Protection and Maximum Availability data protection modes described in Chapter 5, "Data Guard Protection Modes".

- Asynchronous

  The asynchronous redo transport mode transmits redo data asynchronously with respect to transaction commitment. A transaction can commit without waiting for the redo generated by that transaction to be successfully sent to any redo transport destination that uses the asynchronous redo transport mode.

  This transport mode is used by the Maximum Performance data protection mode described in Chapter 5, "Data Guard Protection Modes".

6.2 Configuring Redo Transport Services

This section describes how to configure redo transport services. The following topics are discussed:

- Redo Transport Security
- Configuring an Oracle Database to Send Redo Data
- Configuring an Oracle Database to Receive Redo Data

The section is written at a level of detail that assumes that the reader has a thorough understanding of the following topics, which are described in the Oracle Database Administrator's Guide and the Oracle Database Backup and Recovery User's Guide:

- Database administrator authentication
- Database initialization parameters
- Managing a redo log
- Managing archived redo logs
- Flash recovery areas

6.2.1 Redo Transport Security

Redo transport uses Oracle Net sessions to transport redo data. These redo transport sessions are authenticated using either the Secure Socket Layer (SSL) protocol or a remote login password file.

6.2.1.1 Redo Transport Authentication Using SSL

Secure Sockets Layer (SSL) is an industry standard protocol for securing network connections. SSL uses RSA public key cryptography and symmetric key cryptography to provide authentication, encryption, and data integrity. SSL is automatically used for redo transport authentication between two Oracle databases if:

- The databases are members of the same Oracle Internet Directory (OID) enterprise domain and that domain allows the use of current user database links.
- The `LOG_ARCHIVE_DEST_`*n*, `FAL_SERVER`, and `FAL_CLIENT` database initialization parameters that correspond to the databases use Oracle Net connect descriptors configured for SSL.

- Each database has an Oracle wallet or a supported hardware security module that contains a user certificate with a distinguished name (DN) that matches the DN in the OID entry for the database.

  See Also:

  - *Oracle Database Advanced Security Administrator's Guide* for more information about configuring SSL Authentication
  - *Oracle Database Enterprise User Security Administrator's Guide* for more information about administering enterprise domains
  - *Oracle Internet Directory Administrator's Guide* for information about administering Oracle Internet Directory

6.2.1.2 Redo Transport Authentication Using a Password File

If the SSL authentication requirements are not met, each database must use a remote login password file. In a Data Guard configuration, all physical and snapshot standby databases must use a copy of the password file from the primary database, and that copy must be refreshed whenever the `SYSOPER` or `SYSDBA` privilege is granted or revoked, and after the password of any user with these privileges is changed.

When a password file is used for redo transport authentication, the password of the user account used for redo transport authentication is compared between the database initiating a redo transport session and the target database. The password must be the same at both databases to create a redo transport session.

By default, the password of the `SYS` user is used to authenticate redo transport sessions when a password file is used. The `REDO_TRANSPORT_USER` database initialization parameter can be used to select a different user password for redo transport authentication by setting this parameter to the name of any user who has been granted the `SYSOPER` privilege. For administrative ease, Oracle recommends that the `REDO_TRANSPORT_USER` parameter be set to the same value on the redo source database and at each redo transport destination.

See Also: *Oracle Database Administrator's Guide* for more information creating and maintaining remote login password files

6.2.2 Configuring an Oracle Database to Send Redo Data

This section describes how to configure an Oracle database to send redo data to a redo transport destination.

The `LOG_ARCHIVE_DEST_`*n* database initialization parameter (where *n* is an integer from 1 to 10) is used to specify the location of a local archive redo log or to specify a redo transport destination. This section describes the latter use of this parameter.

There is a `LOG_ARCHIVE_DEST_STATE_`*n* database initialization parameter (where *n* is an integer from 1 to 10) that corresponds to each `LOG_ARCHIVE_DEST_`*n* parameter. This parameter is used to enable or disable the corresponding redo destination. Table 6–1 shows the valid values that can be assigned to this parameter.

Table 6–1 LOG_ARCHIVE_DEST_STATE_n Initialization Parameter Values

Value	Description
`ENABLE`	Redo transport services can transmit redo data to this destination. This is the default.
`DEFER`	Redo transport services will not transmit redo data to this destination.

Table 6–1 (Cont.) LOG_ARCHIVE_DEST_STATE_n Initialization Parameter Values

Value	Description
ALTERNATE	This destination will become enabled if communication to its associated destination fails.

A redo transport destination is configured by setting the LOG_ARCHIVE_DEST_*n* parameter to a character string that includes one or more attributes. This section briefly describes the most commonly used attributes. See Chapter 15 for a full description of all LOG_ARCHIVE_DEST_*n* parameter attributes.

The SERVICE attribute, which is a mandatory attribute for a redo transport destination, must be the first attribute specified in the attribute list. The SERVICE attribute is used to specify the Oracle Net service name used to connect to a redo transport destination. See the *Oracle Database Net Services Administrator's Guide* for information about Oracle Net service names.

The SYNC attribute is used to specify that the synchronous redo transport mode be used to send redo data to a redo transport destination.

The ASYNC attribute is used to specify that the asynchronous redo transport mode be used to send redo data to a redo transport destination. The asynchronous redo transport mode will be used if neither the SYNC nor the ASYNC attribute is specified.

The NET_TIMEOUT attribute is used to specify how long the LGWR process will block waiting for an acknowledgement that redo data has been successfully received by a destination that uses the synchronous redo transport mode. If an acknowledgement is not received within NET_TIMEOUT seconds, the redo transport connection is terminated and an error is logged.

Oracle recommends that the NET_TIMEOUT attribute be specified whenever the synchronous redo transport mode is used, so that the maximum duration of a redo source database stall caused by a redo transport fault can be precisely controlled. See Section 6.3.2 for information about monitoring synchronous redo transport mode response time.

The AFFIRM attribute is used to specify that redo received from a redo source database is not acknowledged until it has been written to the standby redo log. The NOAFFIRM attribute is used to specify that received redo is acknowledged without waiting for received redo to be written to the standby redo log.

The DB_UNIQUE_NAME attribute is used to specify the DB_UNIQUE_NAME of a redo transport destination. The DB_UNIQUE_NAME attribute must be specified if the LOG_ARCHIVE_CONFIG database initialization parameter has been defined and its value includes a DG_CONFIG list.

If the DB_UNIQUE_NAME attribute is specified, its value must match one of the DB_UNIQUE_NAME values in the DG_CONFIG list. It must also match the value of the DB_UNIQUE_NAME database initialization parameter at the redo transport destination. If either match fails, an error is logged and redo transport will not be possible to that destination.

The VALID_FOR attribute is used to specify when redo transport services transmits redo data to a redo transport destination. Oracle recommends that the VALID_FOR attribute be specified for each redo transport destination at every site in a Data Guard configuration so that redo transport services will continue to send redo data to all standby databases after a role transition, regardless of which standby database assumes the primary role.

The REOPEN attribute is used to specify the minimum number of seconds between automatic reconnect attempts to a redo transport destination that is inactive because of a previous error.

The COMPRESSION attribute is used to specify that redo data is transmitted to a redo transport destination in compressed form when resolving redo data gaps. Redo transport compression can significantly improve redo gap resolution time when network links with low bandwidth and high latency are used for redo transport. Redo gap resolution is discussed in Section 6.3.3.

The following example uses all of the LOG_ARCHIVE_DEST_*n* attributes described in this section. Two redo transport destinations are defined and enabled. The first destination uses the asynchronous redo transport mode. The second destination uses the synchronous redo transport mode with a 30-second timeout. A DB_UNIQUE_NAME has been specified for both destinations, as has the use of compression when resolving redo gaps. If a redo transport fault occurs at either destination, redo transport will attempt to reconnect to that destination, but not more frequently than once every 60 seconds.

```
DB_UNIQUE_NAME=BOSTON
LOG_ARCHIVE_CONFIG='DG_CONFIG=(BOSTON,CHICAGO,DENVER)'
LOG_ARCHIVE_DEST_2='SERVICE=CHICAGO ASYNC NOAFFIRM VALID_FOR=(ONLINE_LOGFILE,
PRIMARY_ROLE) REOPEN=60 COMPRESSION=ENABLE  DB_UNIQUE_NAME=CHICAGO'
LOG_ARCHIVE_DEST_STATE_2='ENABLE'
LOG_ARCHIVE_DEST_3='SERVICE=DENVER SYNC AFFIRM NET_TIMEOUT=30 VALID_FOR=(ONLINE_
LOGFILE,PRIMARY_ROLE) REOPEN=60 COMPRESSION=ENABLE  DB_UNIQUE_NAME=DENVER'
LOG_ARCHIVE_DEST_STATE_3='ENABLE'
```

6.2.2.1 Viewing Attributes With V$ARCHIVE_DEST

The V$ARCHIVE_DEST view can be queried to see the current settings and status for each redo transport destination.

6.2.3 Configuring an Oracle Database to Receive Redo Data

This section describes how to configure a redo transport destination to receive and to archive redo data from a redo source database.

The following topics are discussed:

- Creating and Managing a Standby Redo Log
- Configuring Standby Redo Log Archival

6.2.3.1 Creating and Managing a Standby Redo Log

The synchronous and asynchronous redo transport modes require that a redo transport destination have a standby redo log. A standby redo log is used to store redo received from another Oracle database. Standby redo logs are structurally identical to redo logs, and are created and managed using the same SQL statements used to create and manage redo logs.

Redo received from another Oracle database via redo transport is written to the current standby redo log group by a RFS background process. When a log switch occurs on the redo source database, incoming redo is then written to the next standby redo log group, and the previously used standby redo log group is archived by an ARCn background process.

The process of sequentially filling and then archiving redo log file groups at a redo source database is mirrored at each redo transport destination by the sequential filling and archiving of standby redo log groups.

Each standby redo log file must be at least as large as the largest redo log file in the redo log of the redo source database. For administrative ease, Oracle recommends that all redo log files in the redo log at the redo source database and the standby redo log at a redo transport destination be of the same size.

The standby redo log must have at least one more redo log group than the redo log on the redo source database.

Perform the following query on a redo source database to determine the size of each log file and the number of log groups in the redo log:

```
SQL> SELECT GROUP#, BYTES FROM V$LOG;
```

Perform the following query on a redo destination database to determine the size of each log file and the number of log groups in the standby redo log:

```
SQL> SELECT GROUP#, BYTES FROM V$STANDBY_LOG;
```

Oracle recommends that a standby redo log be created on the primary database in a Data Guard configuration so that it is immediately ready to receive redo data following a switchover to the standby role.

The `ALTER DATABASE ADD STANDBY LOGFILE` SQL statement is used to create a standby redo log and to add standby redo log groups to an existing standby redo log.

For example, assume that the redo log on the redo source database has two redo log groups and that each of those contain one 500 MB redo log file. In this case, the standby redo log should have at least 3 standby redo log groups to satisfy the requirement that a standby redo log must have at least one more redo log group than the redo log at the redo source database.

The following SQL statements might be used to create a standby redo log that is appropriate for the previous scenario:

```
ALTER DATABASE ADD STANDBY LOGFILE
   ('/oracle/dbs/slog1.rdo') SIZE 500M;

ALTER DATABASE ADD STANDBY LOGFILE
   ('/oracle/dbs/slog2.rdo') SIZE 500M;

ALTER DATABASE ADD STANDBY LOGFILE
   ('/oracle/dbs/slog3.rdo') SIZE 500M;
```

> **Caution:** Whenever a redo log group is added to the primary database in an Oracle Data Guard configuration, a standby redo log group must also be added to the standby redo log at each standby database in the configuration that uses the synchronous redo transport mode. If this is not done, a primary database that is running in the maximum protection data protection mode may shut down, and a primary database that is running in the maximum availability data protection mode may shift to the maximum performance data protection mode.

6.2.3.2 Configuring Standby Redo Log Archival

This section describes how to configure standby redo log archival.

See Also:

- *Oracle Database Administrator's Guide* for more information about managing archived redo logs
- *Oracle Database Backup and Recovery User's Guide* for more information about flash recovery areas

6.2.3.2.1 Standby Redo Log Archival to a Flash Recovery Area Take the following steps to set up standby redo log archival to a flash recovery area:

1. Set the `LOCATION` attribute of a `LOG_ARCHIVE_DEST_`*n* parameter to `USE_DB_RECOVERY_FILE_DEST`.
2. Set the `VALID_FOR` attribute of the same `LOG_ARCHIVE_DEST_`*n* parameter to a value that allows standby redo log archival.

The following are some sample parameter values that might be used to configure a physical standby database to archive its standby redo log to the flash recovery area:

```
LOG_ARCHIVE_DEST_2 = 'LOCATION=USE_DB_RECOVERY_FILE_DEST
VALID_FOR=(STANDBY_LOGFILE,STANDBY_ROLE)'
LOG_ARCHIVE_DEST_STATE_2=ENABLE
```

Oracle recommends the use of a flash recovery area, because it simplifies the management of archived redo log files.

6.2.3.2.2 Standby Redo Log Archival to a Local FIle System Location Take the following steps to set up standby redo log archival to a local file system location:

1. Set the `LOCATION` attribute of a `LOG_ARCHIVE_DEST_`*n* parameter to a valid pathname.
2. Set the `VALID_FOR` attribute of the same `LOG_ARCHIVE_DEST_`*n* parameter to a value that allows standby redo log archival.

The following are some sample parameter values that might be used to configure a physical standby database to archive its standby redo log to a local file system location:

```
LOG_ARCHIVE_DEST_2 = 'LOCATION = /disk2/archive
VALID_FOR=(STANDBY_LOGFILE,STANDBY_ROLE)'
LOG_ARCHIVE_DEST_STATE_2=ENABLE
```

6.3 Monitoring Redo Transport Services

This section discusses the following topics:

- Monitoring Redo Transport Status
- Monitoring Synchronous Redo Transport Response Time
- Redo Gap Detection and Resolution
- Redo Transport Services Wait Events

6.3.1 Monitoring Redo Transport Status

This section describes the steps used to monitor redo transport status on a redo source database.

Step 1 Determine the most recently archived redo log file.

Perform the following query on the redo source database to determine the most recently archived sequence number for each thread:

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

Step 2 Determine the most recently archived redo log file at each redo transport destination.

Perform the following query on the redo source database to determine the most recently archived redo log file at each redo transport destination:

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

Step 3 Find out if archived redo log files have been received at a redo transport destination.

A query can be performed at a redo source database to find out if an archived redo log file has been received at a particular redo transport destination. Each destination has an ID number associated with it. You can query the `DEST_ID` column of the `V$ARCHIVE_DEST` view on a database to identify each destination's ID number.

Assume that destination 1 points to the local archived redo log and that destination 2 points to a redo transport destination. Perform the following query at the redo source database to find out if any log files are missing at the redo transport destination:

```
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
```

Step 4 Trace the progression of redo transmitted to a redo transport destination.

Set the `LOG_ARCHIVE_TRACE` database initialization parameter at a redo source database and at each redo transport destination to trace redo transport progress. See Appendix G for complete details and examples.

6.3.2 Monitoring Synchronous Redo Transport Response Time

The `V$REDO_DEST_RESP_HISTOGRAM` view contains response time data for each redo transport destination. This response time data is maintained for redo transport messages sent via the synchronous redo transport mode.

The data for each destination consists of a series of rows, with one row for each response time. To simplify record keeping, response times are rounded up to the nearest whole second for response times less than 300 seconds. Response times greater than 300 seconds are round up to 600, 1200, 2400, 4800, or 9600 seconds.

Each row contains four columns: FREQUENCY, DURATION, DEST_ID, and TIME.

The FREQUENCY column contains the number of times that a given response time has been observed. The DURATION column corresponds to the response time. The DEST_ID column identifies the destination. The TIME column contains a timestamp taken when the row was last updated.

The response time data in this view is useful for identifying synchronous redo transport mode performance issues that can affect transaction throughput on a redo source database. It is also useful for tuning the NET_TIMEOUT attribute.

The next three examples show example queries for destination 2, which corresponds to the LOG_ARCHIVE_DEST_2 parameter. To display response time data for a different destination, simply change the DEST_ID in the query.

Perform the following query on a redo source database to display the response time histogram for destination 2:

```
SQL> SELECT FREQUENCY, DURATION FROM
2> V$REDO_DEST_RESP_HISTOGRAM WHERE DEST_ID=2 AND FREQUENCY>1;
```

Perform the following query on a redo source database to display the slowest response time for destination 2:

```
SQL> SELECT max(DURATION) FROM V$REDO_DEST_RESP_HISTOGRAM
2> WHERE DEST_ID=2 AND FREQUENCY>1;
```

Perform the following query on a redo source database to display the fastest response time for destination 2:

```
SQL> SELECT min( DURATION) FROM V$REDO_DEST_RESP_HISTOGRAM
2> WHERE DEST_ID=2 AND FREQUENCY>1;
```

Note: The highest observed response time for a destination cannot exceed the highest specified NET_TIMEOUT value specified for that destination, because synchronous redo transport mode sessions are terminated if a redo transport destination does not respond to a redo transport message within NET_TIMEOUT seconds.

6.3.3 Redo Gap Detection and Resolution

A redo gap occurs whenever redo transmission is interrupted. When redo transmission resumes, redo transport services automatically detects the redo gap and resolves it by sending the missing redo to the destination.

The time needed to resolve a redo gap is directly proportional to the size of the gap and inversely proportional to the effective throughput of the network link between the redo source database and the redo transport destination. Redo transport services has two options that may reduce redo gap resolution time when low performance network links are used:

- Redo Transport Compression

  The COMPRESSION attribute of the LOG_ARCHIVE_DEST_*n* parameter can be used to specify that redo transport compression be used to compress the redo sent to resolve a redo gap.

- Parallel Redo Transport Network Sessions

  The MAX_CONNECTIONS attribute of the LOG_ARCHIVE_DEST_*n* parameter can be used to specify that more than one network session be used to send the redo needed to resolve a redo gap.

See Chapter 15, "LOG_ARCHIVE_DEST_n Parameter Attributes" for more information about the COMPRESSION and MAX_CONNECTIONS attributes.

6.3.3.1 Manual Gap Resolution

In some situations, gap resolution cannot be performed automatically and it must be performed manually. For example, redo gap resolution must be performed manually on a logical standby database if the primary database is unavailable.

Perform the following query at the physical standby database to determine if there is redo gap on a physical standby database:

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

The output from the previous example indicates that the physical standby database is currently missing log files from sequence 7 to sequence 10 for thread 1.

Perform the following query on the primary database to locate the archived redo log files on the 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
2> DEST_ID=1 AND 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
```

> **Note:** This query may return consecutive sequences for a given thread. In that case, there is no actual gap, but the associated thread was disabled and enabled within the time period of generating these two archived logs. The query also does not identify the gap that may exist at the tail end for a given thread. For instance, if the primary database has generated archived logs up to sequence 100 for thread 1, and the latest archived log that the logical standby database has received for the given thread is the one associated with sequence 77, this query will not return any rows, although we have a gap for the archived logs associated with sequences 78 to 100.

Copy these log files to the physical standby database and register them using the ALTER DATABASE REGISTER LOGFILE. 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';
SQL> ALTER DATABASE REGISTER LOGFILE
'/physical_standby1/thread1_dest/arcr_1_9.arc';
```

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

To determine if there is a redo gap on a logical standby database, 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. For example:

```
SQL> ALTER DATABASE REGISTER LOGICAL LOGFILE '/disk1/oracle/dbs/log-1292880008_
7.arc';
SQL> ALTER DATABASE REGISTER LOGICAL LOGFILE '/disk1/oracle/dbs/log-1292880008_
8.arc';
SQL> ALTER DATABASE REGISTER LOGICAL LOGFILE '/disk1/oracle/dbs/log-1292880008_
9.arc';
```

Note: A query based on the DBA_LOGSTDBY_LOG view on a logical standby database, as specified above, only returns the gap that is currently blocking SQL Apply from continuing. After resolving the gap, query the DBA_LOGSTDBY_LOG view again on the logical standby database to determine if there is another gap sequence. Repeat this process until there are no more gaps.

6.3.4 Redo Transport Services Wait Events

Table 6–2 lists several of the Oracle wait events used to track redo transport wait time on a redo source database. These wait events are found in the V$SYSTEM_EVENT dynamic performance view.

For a complete list of the Oracle wait events used by redo transport, see the Oracle Data Guard Redo Transport and Network Best Practices white paper on the Oracle Maximum Availability Architecture (MAA) home page at:

http://otn.oracle.com/deploy/availability/htdocs/maa.htm

Table 6–2 Redo Transport Wait Events

Wait Event	Description
LNS wait on ATTACH	Total time spent waiting for redo transport sessions to be established to all ASYNC and SYNC redo transport destinations
LNS wait on SENDREQ	Total time spent waiting for redo data to be written to all ASYNC and SYNC redo transport destinations
LNS wait on DETACH	Total time spent waiting for redo transport connections to be terminated to all ASYNC and SYNC redo transport destinations

6.4 Tuning Redo Transport

The Oracle Data Guard Redo Transport and Network Configuration Best Practices white paper describes how to optimize redo transport for best performance. This paper is available on the Oracle Maximum Availability Architecture (MAA) home page at:

http://otn.oracle.com/deploy/availability/htdocs/maa.htm

7

Apply Services

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

7.1 Introduction to Apply Services

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, apply services wait for the *full* archived redo log file to arrive on the standby database before applying it to the standby database. However, if you use a standby redo log, you can enable **real-time apply**, which allows Data Guard to recover redo data from the current standby redo log file as it is being filled. Real-time apply is described in more detail in Section 7.2.1.

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.

- SQL Apply (logical standby databases only)

  Reconstitutes SQL statements from the redo received from the primary database and executes the SQL statements against the logical standby database.

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

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

7.2 Apply Services Configuration Options

This section contains the following topics:

7.2.1 Using Real-Time Apply to Apply Redo Data Immediately

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

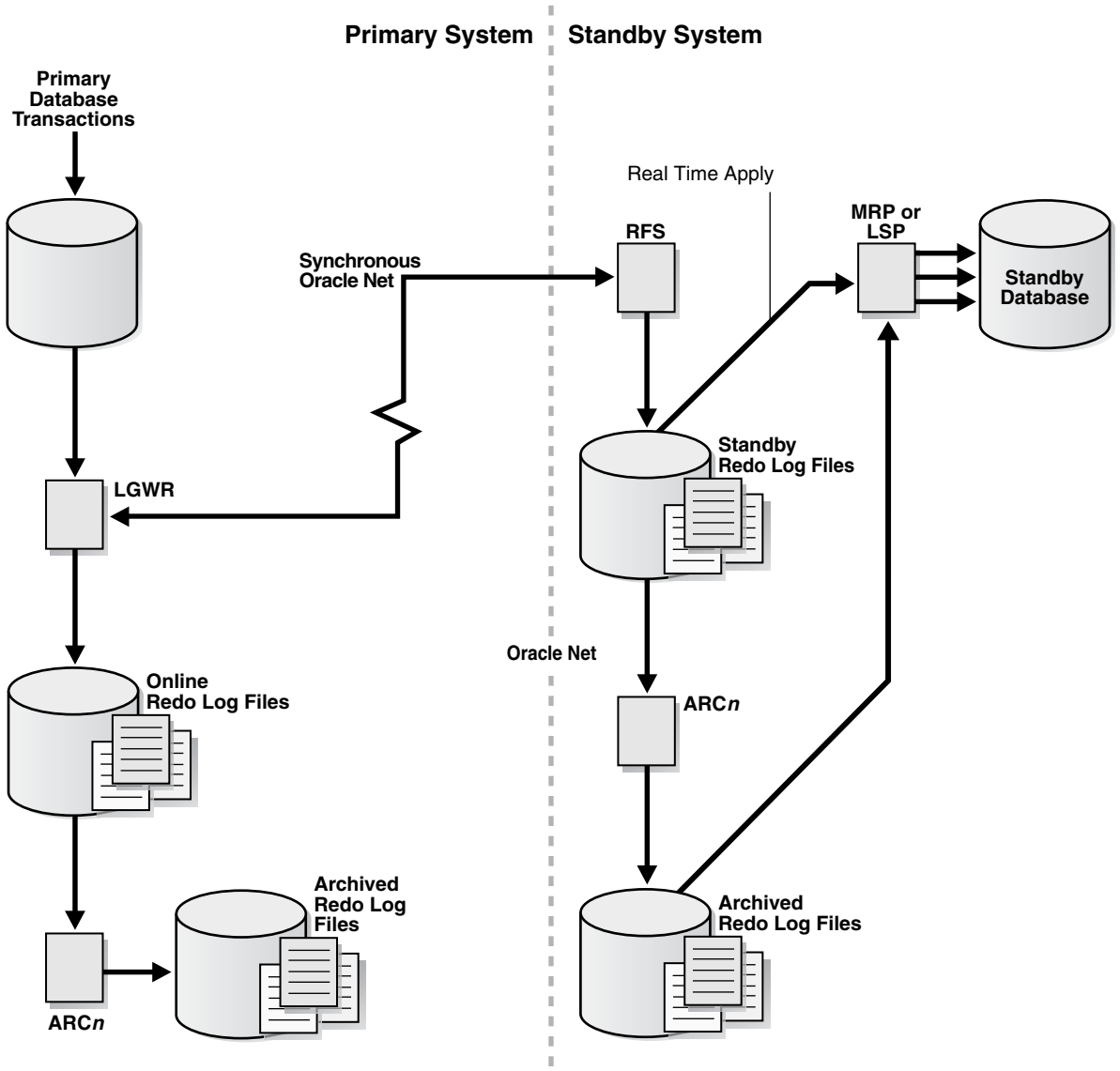
Use the `ALTER DATABASE` statement to enable the real-time apply feature, as follows:

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

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

Figure 7–1 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, apply services can recover redo from standby redo log files as they are being filled.

Figure 7–1 Applying Redo Data to a Standby Destination Using Real-Time Apply



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

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

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

Specifying a Time Delay

You can set a time delay on primary and standby databases using the `DELAY=`*`minutes`* attribute of the `LOG_ARCHIVE_DEST_`*`n`* initialization parameter to

delay applying archived redo log files to the standby database. By default, there is no time delay. If you specify the DELAY attribute without specifying a value, then the default delay interval is 30 minutes.

Canceling a Time Delay

You can cancel a specified delay interval as follows:

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

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

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

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

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

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

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

See Chapter 13 for scenarios showing how to use Data Guard with Flashback Database, and *Oracle Database Backup and Recovery User's Guide* for more information about enabling and using Flashback Database.

7.3 Applying Redo Data to Physical Standby Databases

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

This section contains the following topics:

- Starting Redo Apply
- Stopping Redo Apply
- Monitoring Redo Apply on Physical Standby Databases

7.3.1 Starting Redo Apply

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

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

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

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

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

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

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

  This statement starts a detached server process and immediately returns control to the user. While the managed recovery process is performing recovery in the background, the foreground process that issued the RECOVER statement can continue performing other tasks. This does not disconnect the current SQL session.

- To start real-time apply, include the USING CURRENT LOGFILE clause on the SQL statement. For example:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE USING CURRENT LOGFILE;
```

7.3.2 Stopping Redo Apply

To stop Redo Apply, issue the following SQL statement in another window:

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

7.3.3 Monitoring Redo Apply on Physical Standby Databases

To monitor the status of apply services on a physical standby database, see Section 9.5.1. You can also monitor the standby database using Oracle Enterprise Manager. Also, see the *Oracle Database Reference* for complete reference information about views.

7.4 Applying Redo Data to Logical Standby Databases

SQL Apply converts the data from the archived redo log or standby redo log in to SQL statements and then executes these SQL statements on the logical standby database. Because the logical standby database remains open, tables that are maintained can be used simultaneously for other tasks such as reporting, summations, and queries.

This section contains the following topics:

- Starting SQL Apply
- Stopping SQL Apply on a Logical Standby Database
- Monitoring SQL Apply on Logical Standby Databases

7.4.1 Starting SQL Apply

To start SQL Apply, start the logical standby database and issue the following statement:

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

To start real-time apply on the logical standby database to immediately apply redo data from the standby redo log files on the logical standby database, include the IMMEDIATE keyword as shown in the following statement:

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

7.4.2 Stopping SQL Apply on a Logical Standby Database

To stop SQL Apply, issue the following statement on the logical standby database:

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

When you issue this statement, SQL Apply waits until it has committed all complete transactions that were in the process of being applied. Thus, this command may not stop the SQL Apply processes immediately.

7.4.3 Monitoring SQL Apply on Logical Standby Databases

To monitor SQL Apply, see Section 10.3. You can also monitor the standby database using Oracle Enterprise Manager. See Appendix A, "Troubleshooting Data Guard" and *Oracle Data Guard Broker*.

8

Role Transitions

A Data Guard configuration consists of one database that functions in the primary role and one or more databases that function in the standby role. Typically, the role of each database does not change. However, if Data Guard is used to maintain service in response to a primary database outage, you must initiate a role transition between the current primary database and one standby database in the configuration. To see the current role of the databases, query the `DATABASE_ROLE` column in the `V$DATABASE` view.

The number, location, and type of standby databases in a Data Guard configuration and the way in which redo data from the primary database is propagated to each standby database determine the role-management options available to you in response to a primary database outage.

This chapter describes how to manage role transitions in a Data Guard configuration. It contains the following topics:

The role transitions described in this chapter are invoked manually using SQL statements. You can also use the Oracle Data Guard broker to simplify role transitions and automate failovers.

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

8.1 Introduction to Role Transitions

A database operates in one of the following mutually exclusive roles: **primary** or **standby**. Data Guard enables you to change these roles dynamically by issuing the

SQL statements described in this chapter, or by using either of the Data Guard broker's interfaces. Oracle Data Guard supports the following role transitions:

- **Switchover**

  Allows the primary database to switch roles with one of its standby databases. There is no data loss during a switchover. After a switchover, each database continues to participate in the Data Guard configuration with its new role.

- **Failover**

  Changes a standby database to the primary role in response to a primary database failure. If the primary database was not operating in either maximum protection mode or maximum availability mode before the failure, some data loss may occur. If Flashback Database is enabled on the primary database, it can be reinstated as a standby for the new primary database once the reason for the failure is corrected.

Section 8.1.1, "Preparing for a Role Transition" on page 8-2 helps you choose the role transition that best minimizes downtime and risk of data loss. Switchovers and failovers are described in more detail in Section 8.1.3, "Switchovers" on page 8-4 and Section 8.1.4, "Failovers" on page 8-6, respectively.

8.1.1 Preparing for a Role Transition

Before starting any role transition, perform the following preparations:

- Verify that each database is properly configured for the role that it is about to assume. See Chapter 3, "Creating a Physical Standby Database" and Chapter 4, "Creating a Logical Standby Database" for information about how to configure database initialization parameters, archivelog mode, standby redo logs, and online redo logs on primary and standby databases.

  Note: You must define the `LOG_ARCHIVE_DEST_`*n* and `LOG_ARCHIVE_DEST_STATE_`*n* parameters on each standby database so that when a switchover or failover occurs, all standby sites continue to receive redo data from the new primary database.

- Ensure temporary files exist on the standby database that match the temporary files on the primary database.
- Remove any delay in applying redo that may be in effect on the standby database that will become the new primary database.
- Before performing a switchover from an Oracle RAC primary database to a physical standby database, shut down all but one primary database instance. Any primary database instances shut down at this time can be started after the switchover completes.

  Before performing a switchover or a failover to an Oracle RAC physical standby database, shut down all but one standby database instance. Any standby database instances shut down at this time can be restarted after the role transition completes.

8.1.2 Choosing a Target Standby Database for a Role Transition

For a Data Guard configuration with multiple standby databases, there are a number of factors to consider when choosing the target standby database for a role transition. These include the following:

- Locality of the standby database.
- The capability of the standby database (hardware specifications—such as the number of CPUs, I/O bandwidth available, and so on).
- The time it will take to perform the role transition. This is affected by how far behind the standby database is in terms of application of redo data, and how much flexibility you have in terms of trading off application availability with data loss.
- Standby database type.

The type of standby chosen as the role transition target determines how other standby databases in the configuration will behave after the role transition. If the new primary was a physical standby before the role transition, all other standby databases in the configuration will become standbys of the new primary. If the new primary was a logical standby before the role transition, then all other logical standbys in the configuration will become standbys of the new primary, but physical standbys in the configuration will continue to be standbys of the old primary and will therefore not protect the new primary. In the latter case, a future switchover or failover back to the original primary database will return all standbys to their original role as standbys of the current primary. For the reasons described above, a physical standby is generally the best role transition target in a configuration that contains both physical and logical standbys.

Note: A snapshot standby cannot be the target of a role transition.

Data Guard provides the V$DATAGUARD_STATS view that can be used to evaluate each standby database in terms of the currency of the data in the standby database, and the time it will take to perform a role transition if all available redo data is applied to the standby database. For example:

```
SQL> COLUMN NAME FORMAT A18
SQL> COLUMN VALUE FORMAT A16
SQL> COLUMN TIME_COMPUTED FORMAT A24
SQL> SELECT * FROM V$DATAGUARD_STATS;
NAME               VALUE             TIME_COMPUTED
------------------ ----------------  ------------------------
apply finish time  +00 00:00:02.4    15-MAY-2005 10:32:49
       second(1)
       interval
apply lag          +00 0:00:04       15-MAY-2005 10:32:49
       second(0)
       interval
transport lag      +00 00:00:00      15-MAY-2005 10:32:49
       second(0)
       interval
```

The time at which each of the statistics is computed is shown in the TIME_COMPUTED column. The V$DATATGUARD_STATS.TIME_COMPUTED column is a timestamp taken when the metric in a V$DATATGUARD_STATS row is computed. This column indicates the *freshness* of the associated metric. This shows that for this standby database, there is no transport lag, that apply services has not applied the redo generated in the last 4 seconds (apply lag), and that it will take apply services 2.4 seconds to finish applying the unapplied redo (apply finish time). The APPLY LAG and TRANSPORT LAG metrics are computed based on information received from the primary database, and these metrics become stale if communications between the primary and standby database are disrupted. An unchanging value in this column for

the APPLY LAG and TRANSPORT LAG metrics indicates that these metrics are not being updated (or have become stale), possibly due to a communications fault between the primary and standby databases.

8.1.3 Switchovers

A switchover is typically used to reduce primary database downtime during planned outages, such as operating system or hardware upgrades, or rolling upgrades of the Oracle database software and patch sets (described in Chapter 12, "Using SQL Apply to Upgrade the Oracle Database").

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

Figure 8–1 shows a two-site Data Guard configuration before the roles of the databases are switched. The primary database is in San Francisco, and the standby database is in Boston.

Figure 8–1 Data Guard Configuration Before Switchover

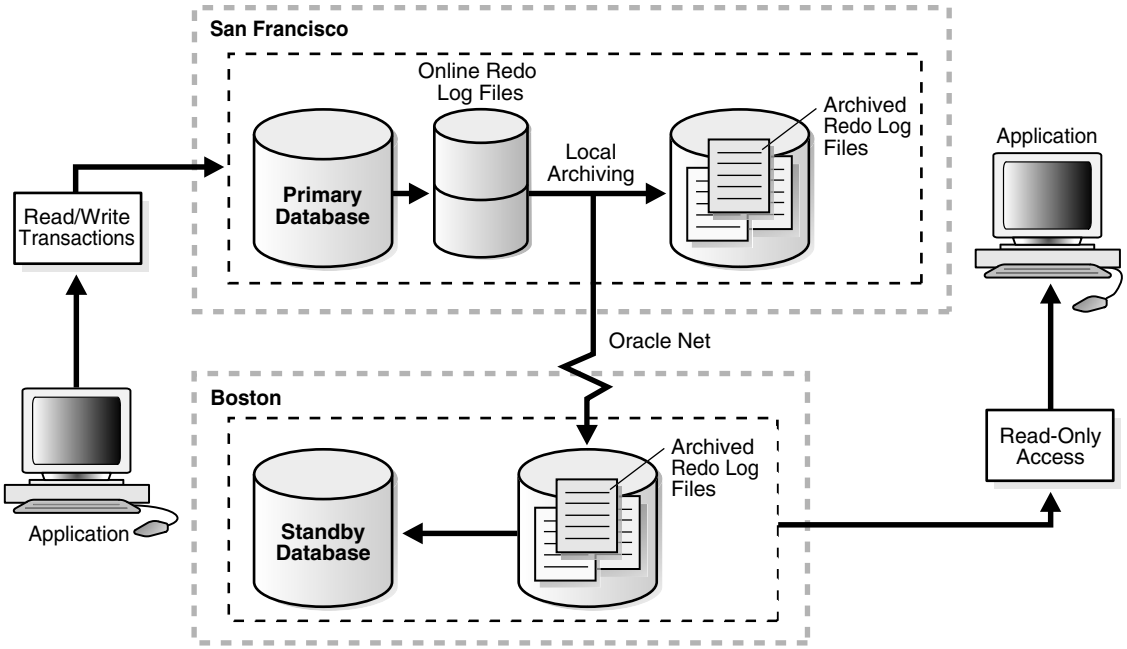


Figure 8–2 shows the Data Guard environment after the original primary database was switched over to a standby database, but before the original standby database has become the new primary database. At this stage, the Data Guard configuration temporarily has two standby databases.

Figure 8–2 Standby Databases Before Switchover to the New Primary Database

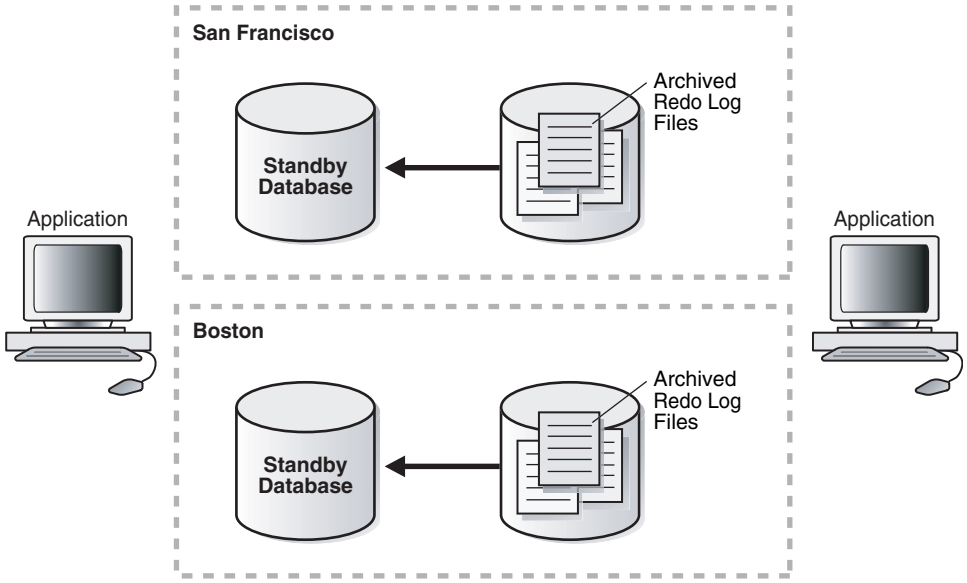
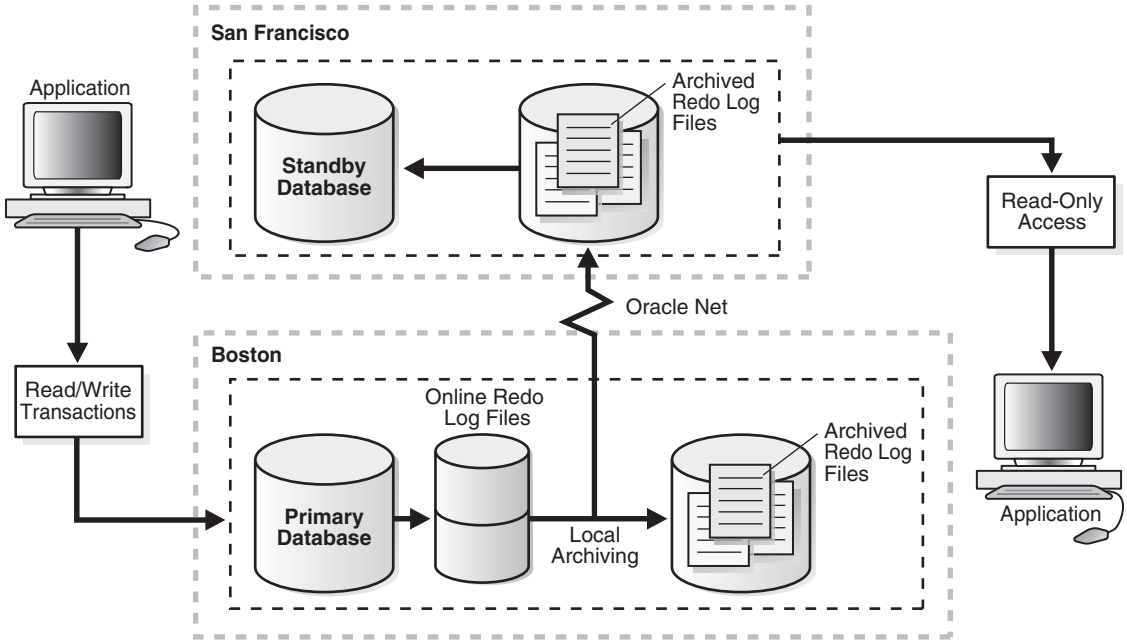


Figure 8–3 shows the Data Guard environment after a switchover took place. The original standby database became the new primary database. The primary database is now in Boston, and the standby database is now in San Francisco.

Figure 8–3 Data Guard Environment After Switchover



Preparing for a Switchover

Ensure the prerequisites listed in Section 8.1.1 are satisfied. In addition, the following prerequisites must be met for a switchover:

- For switchovers involving a physical standby database, verify that the primary database is open and that redo apply is active on the standby database. See

Section 7.3, "Applying Redo Data to Physical Standby Databases" for more information about Redo Apply.

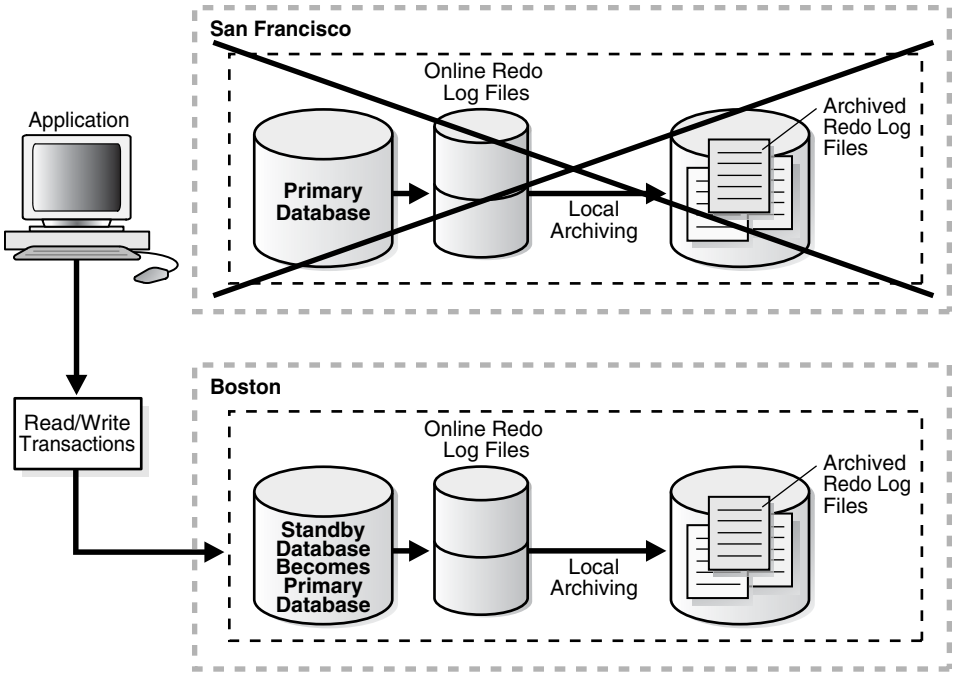
- For switchovers involving a logical standby database, verify both the primary and standby database instances are open and that SQL Apply is active. See Section 7.4, "Applying Redo Data to Logical Standby Databases" for more information about SQL Apply.

8.1.4 Failovers

A failover is typically used only when the primary database becomes unavailable, and there is no possibility of restoring it to service within a reasonable period of time. The specific actions performed during a failover vary based on whether a logical or a physical standby database is involved in the failover, the state of the Data Guard configuration at the time of the failover, and on the specific SQL statements used to initiate the failover.

Figure 8–4 shows the result of a failover from a primary database in San Francisco to a physical standby database in Boston.

Figure 8–4 Failover to a Standby Database



Preparing for a Failover

If possible, before performing a failover, you should transfer as much of the available and unapplied primary database redo data as possible to the standby database.

Ensure the prerequisites listed in Section 8.1.1, "Preparing for a Role Transition" on page 8-2 are satisfied. In addition, the following prerequisites must be met for a failover:

- If a standby database currently running in maximum protection mode will be involved in the failover, first place it in maximum performance mode by issuing the following statement on the standby database:

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

Then, if appropriate standby databases are available, you can reset the desired protection mode on the new primary database after the failover completes.

This is required because you cannot fail over to a standby database that is in maximum protection mode. In addition, if a primary database in maximum protection mode is still actively communicating with the standby database, issuing the `ALTER DATABASE` statement to change the standby database from maximum protection mode to maximum performance mode will not succeed. Because a failover removes the original primary database from the Data Guard configuration, these features serve to protect a primary database operating in maximum protection mode from the effects of an unintended failover.

Note: Do not fail over to a standby database to test whether or not the standby database is being updated correctly. Instead:

- See Section 3.2.7, "Verify the Physical Standby Database Is Performing Properly"
- See Section 4.2.6, "Verify the Logical Standby Database Is Performing Properly"

8.1.5 Role Transition Triggers

The `DB_ROLE_CHANGE` system event is signaled whenever a role transition occurs. This system event is signaled immediately if the database is open when the role transition occurs, or the next time the database is opened if it is closed when a role transition occurs.

The `DB_ROLE_CHANGE` system event can be used to fire a trigger that performs a set of actions whenever a role transition occurs.

8.2 Role Transitions Involving Physical Standby Databases

This section describes how to perform a switchover or failover to a physical standby database.

8.2.1 Performing a Switchover to a Physical Standby Database

This section describes how to perform a switchover to a physical standby database.

A switchover is initiated on the primary database and is completed on the target standby database.

Step 1 Verify that the primary database can be switched to the standby role.

Query the `SWITCHOVER_STATUS` column of the `V$DATABASE` view on the primary database.

For example:

```
SQL> SELECT SWITCHOVER_STATUS FROM V$DATABASE;
SWITCHOVER_STATUS
 -----------------
 TO STANDBY
 1 row selected
```

A value of `TO STANDBY` or `SESSIONS ACTIVE` indicates that the primary database can be switched to the standby role. If neither of these values is returned, a switchover

is not possible because redo transport is either misconfigured or is not functioning properly. See Chapter 6 for information about configuring and monitoring redo transport.

Step 2 Initiate the switchover on the primary database.

Issue the following SQL statement on the primary database to switch it to the standby role:

```
SQL> ALTER DATABASE COMMIT TO SWITCHOVER TO PHYSICAL STANDBY WITH
SESSION SHUTDOWN;
```

This statement converts the primary database into a physical 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.

> **Note:** The `WITH SESSION SHUTDOWN` clause can be omitted from the switchover statement if the query performed in the previous step returned `TO STANDBY`.

Step 3 Shut down and then mount the former primary database.

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

At this point in the switchover process, the original primary database is a physical standby database (see Figure 8–2).

Step 4 Verify that the switchover target is ready to be switched to the primary role.

Query the `SWITCHOVER_STATUS` column of the `V$DATABASE` view on the standby database.

For example:

```
SQL> SELECT SWITCHOVER_STATUS FROM V$DATABASE;
SWITCHOVER_STATUS
-----------------
TO_PRIMARY
1 row selected
```

A value of `TO PRIMARY` or `SESSIONS ACTIVE` indicates that the standby database is ready to be switched to the primary role. If neither of these values is returned, verify that redo apply is active and that redo transport is configured and working properly. Continue to query this column until the value returned is either `TO PRIMARY` or `SESSIONS ACTIVE`.

Step 5 Switch the target physical standby database role to the primary role.

Issue the following SQL statement on the target physical standby database:

```
SQL> ALTER DATABASE COMMIT TO SWITCHOVER TO PRIMARY WITH SESSION SHUTDOWN;
```

> **Note:** The `WITH SESSION SHUTDOWN` clause can be omitted from the switchover statement if the query performed in the previous step returned `TO PRIMARY`.

Step 6 Open the new primary database.

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

Step 7 Start redo apply on the new physical standby database.

For example:

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

8.2.2 Performing a Failover to a Physical Standby Database

Step 1 Identify and resolve any redo gaps.

Query the V$ARCHIVE_GAP view to determine if there are any redo gaps on the target standby database.

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 any 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 a gap, you must repeat the query until no more rows are returned.

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
```

If possible, copy any archived redo log files from the primary database that have 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';
```

If any missing archived redo log files are copied to the target standby database, go back to Step 1 to verify that no additional gaps have been introduced.

If, after performing Step 1 through Step 3, you are not able to resolve all 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 Stop Redo Apply.

Issue the following SQL statement:

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

Step 5 Finish applying all received redo data.

Issue the following SQL statement:

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

If this statement completes without error, proceed to Step 6.

If an error occurs, some received redo data was not applied. Try to resolve the cause of the error and re-issue the statement before proceeding to the next step.

If the error condition cannot be resolved, a failover can still be performed (with some data loss) by issuing the following SQL statement:

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

Proceed to Step 8 when the `ACTIVATE` statement completes.

Step 6 Verify that the target standby database is ready to become a primary database.

Query the `SWITCHOVER_STATUS` column of the `V$DATABASE` view on the target standby database.

For example:

```
SQL> SELECT SWITCHOVER_STATUS FROM V$DATABASE;
SWITCHOVER_STATUS
-----------------
TO PRIMARY
1 row selected
```

A value of either `TO PRIMARY` or `SESSIONS ACTIVE` indicates that the standby database is ready to be switched to the primary role. If neither of these values is returned, verify that redo apply is active and continue to query this view until either `TO PRIMARY` or `SESSIONS ACTIVE` is returned.

Step 7 Switch the physical standby database to the primary role.

Issue the following SQL statement:

```
SQL> ALTER DATABASE COMMIT TO SWITCHOVER TO PRIMARY WITH SESSION SHUTDOWN;
```

> **Note:** The `WITH SESSION SHUTDOWN` clause can be omitted from the switchover statement if the query of the `SWITCHOVER_STATUS` column performed in the previous step returned `TO PRIMARY`.

Step 8 Open the new primary database.

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

Step 9 Back up the new primary database.

Oracle recommends that a full backup be taken of the new primary database.

Step 10 Optionally, restore the failed primary database.

After a failover, the original primary database can be converted into a physical standby database of the new primary database using the method described in Section 13.2 or Section 13.7, or it can be re-created as a physical standby database from a backup of the new primary database using the method described in Section 3.2.

Once the original primary database is running in the standby role, a switchover can be performed to restore it to the primary role.

8.3 Role Transitions Involving Logical Standby Databases

This section describes how to perform switchovers and failovers involving a logical standby database.

8.3.1 Performing a Switchover to a Logical Standby Database

When you perform a switchover that changes roles between a primary database and a logical standby database, always initiate the switchover on the primary database and complete it on the logical standby database. These steps must be performed in the order in which they are described or the switchover will not succeed.

Step 1 Verify it is possible to perform a switchover on the primary database.

On the current primary database, query the `SWITCHOVER_STATUS` column of the `V$DATABASE` fixed view on the primary database to verify it is possible to perform a switchover.

For example:

```
SQL> SELECT SWITCHOVER_STATUS FROM V$DATABASE;
SWITCHOVER_STATUS
-----------------
TO STANDBY
1 row selected
```

A value of `TO STANDBY` or `SESSIONS ACTIVE` in the `SWITCHOVER_STATUS` column indicates that it is possible to switch the primary database to the logical standby role. If one of these values is not displayed, then verify the Data Guard configuration is functioning correctly (for example, verify all `LOG_ARCHIVE_DEST_`*n* parameter values are specified correctly). See *Oracle Database Reference* for information about other valid values for the `SWITCHOVER_STATUS` column of the `V$DATABASE` view.

Step 2 Prepare the current primary database for the switchover.

To prepare the current primary database for a logical standby database role, issue the following SQL statement on the primary database:

```
SQL> ALTER DATABASE PREPARE TO SWITCHOVER TO LOGICAL STANDBY;
```

This statement notifies the current primary database that it will soon switch to the logical standby role and begin receiving redo data from a new primary database. You perform this step on the primary database in preparation to receive the LogMiner dictionary to be recorded in the redo stream of the current logical standby database, as described in step 3.

The value PREPARING SWITCHOVER is displayed in the V$DATABASE.SWITCHOVER_STATUS column if this operation succeeds.

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

Use the following statement to build a LogMiner dictionary on the logical standby database that is the target of the switchover:

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

This statement also starts redo transport services on the logical standby database that begins transmitting its redo data to the current primary database and to other standby databases in the Data Guard configuration. The sites receiving redo data from this logical standby database accept the redo data but they do not apply it.

Depending on the work to be done and the size of the database, the switchover can take some time to complete.

The V$DATABASE.SWITCHOVER_STATUS on the logical standby database initially shows PREPARING DICTIONARY while the LogMiner dictionary is being recorded in the redo stream. Once this has completed successfully, the SWITCHOVER_STATUS column shows PREPARING SWITCHOVER.

Step 4 Ensure the current primary database is ready for the future primary database's redo stream.

Before you can complete the role transition of the primary database to the logical standby role, verify the LogMiner dictionary was received by the primary database by querying the SWITCHOVER_STATUS column of the V$DATABASE fixed view on the primary database. Without the receipt of the LogMiner dictionary, the switchover cannot proceed, because the current primary database will not be able to interpret the redo records sent from the future primary database. The SWITCHOVER_STATUS column shows the progress of the switchover.

When the query returns the TO LOGICAL STANDBY value, you can proceed with Step 5. For example:

```
SQL> SELECT SWITCHOVER_STATUS FROM V$DATABASE;
SWITCHOVER_STATUS
-----------------
TO LOGICAL STANDBY
1 row selected
```

Note: You can cancel the switchover operation by issuing the following statements in the order shown:

1. Cancel switchover on the primary database:

   ```
   SQL> ALTER DATABASE PREPARE TO SWITCHOVER CANCEL;
   ```

2. Cancel the switchover on the logical standby database:

   ```
   SQL> ALTER DATABASE PREPARE TO SWITCHOVER CANCEL;
   ```

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

To complete the role transition of the primary database to a logical standby database, issue the following SQL statement:

```
SQL> ALTER DATABASE COMMIT TO SWITCHOVER TO LOGICAL STANDBY;
```

This statement waits for all current transactions on the primary database to end and prevents any new users from starting new transactions, and establishes a point in time where the switchover will be committed.

Executing this statement will also prevent users from making any changes to the data being maintained in the logical standby database. To ensure faster execution, ensure the primary database is in a quiet state with no update activity before issuing the switchover statement (for example, have all users temporarily log off the primary database). You can query the V$TRANSACTION view for information about the status of any current in-progress transactions that could delay execution of this statement.

The primary database has now undergone a role transition to run in the standby database role.

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

Step 6 Ensure all available redo has been applied to the target logical standby database that is about to become the new primary database.

After you complete the role transition of the primary database to the logical standby role and the switchover notification is received by the standby databases in the configuration, you should verify the switchover notification was processed by the target standby database by querying the SWITCHOVER_STATUS column of the V$DATABASE fixed view on the target standby database. Once all available redo records are applied to the logical standby database, SQL Apply automatically shuts down in anticipation of the expected role transition.

The SWITCHOVER_STATUS value is updated to show progress during the switchover. When the status is TO PRIMARY, you can proceed with Step 7.

For example:

```
SQL> SELECT SWITCHOVER_STATUS FROM V$DATABASE;
SWITCHOVER_STATUS
-----------------
TO PRIMARY
1 row selected
```

See *Oracle Database Reference* for information about other valid values for the SWITCHOVER_STATUS column of the V$DATABASE view.

Step 7 Switch the target logical standby database to the primary database role.

On the logical standby database that you want to switch to the primary role, use the following SQL statement to switch the logical standby database to the primary role:

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

There is no need to shut down and restart any logical standby databases that are in the Data Guard configuration. As described in Section 8.1.2 on page 8-2, all other logical standbys in the configuration will become standbys of the new primary, but any physical standby databases will remain standbys of the original primary database.

Step 8 Start SQL Apply on the new logical standby database.

On the new logical standby database, start SQL Apply:

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

8.3.2 Performing a Failover to a Logical Standby Database

This section describes how to perform failovers involving a logical standby database. A failover role transition involving a logical standby database necessitates taking corrective actions on the failed primary database and on all bystander logical standby databases. If Flashback Database was not enabled on the failed primary database, you must re-create the database from backups taken from the current primary database. Otherwise, you can follow the procedure described in Section 13.2 to convert a failed primary database to be a logical standby database for the new primary database.

Depending on the protection mode for the configuration and the attributes you chose for redo transport services, it might be possible to automatically recover all or some of the primary database modifications.

Step 1 Copy and register any missing archived redo log files to the target logical standby database slated to become the new primary database.

Depending on the condition of the components in the configuration, you might have access to the archived redo log files on the primary database. If so, do the following:

1. Determine if any archived redo log files are missing on the logical standby database.
2. Copy missing log files from the primary database to the logical standby database.
3. Register the copied log files.

You can register an archived redo log files with the logical standby database by issuing the following statement. For example:

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

Step 2 Enable remote destinations.

If you have not previously configured role-based destinations, 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.

Step 3 Activate the new primary database.

Issue the following statement on the target logical standby database (that you are transitioning to the new primary role):

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

This statement stops the RFS process, applies remaining redo data in the standby redo log file before the logical standby database becomes a primary database, stops SQL Apply, and activates the database in the primary database role.

If the FINISH APPLY clause is not specified, then unapplied redo from the current standby redo log file will not be applied before the standby database becomes the primary database.

Step 4 Recovering other standby databases after a failover

Follow the method described in Section 13.1 to ensure existing logical standby databases can continue to provide protection for the new primary database.

Step 5 Back up the new primary database.

Back up the new primary database immediately after the Data Guard database failover. Immediately performing a backup is a necessary safety measure, because you cannot recover changes made after the failover without a complete backup copy of the database.

Step 6 Restore the failed primary database.

After a failover, the original primary database can be converted into a logical standby database of the new primary database using the method described in Section 13.2, or it can be recreated as a logical standby database from a backup of the new primary database as described in Chapter 4.

Once the original primary database has been converted into a standby database, a switchover can be performed to restore it to the primary role.

8.4 Using Flashback Database After a Role Transition

After a role transition, you can optionally use the FLASHBACK DATABASE command to revert the databases to a point in time or system change number (SCN) prior to when the role transition occurred.

In a physical standby database environment, you may need to flash back the primary database and all standby databases to maintain the Data Guard configuration. If you flash back the primary database to a certain SCN or time, you must flash back all the standby databases to either the same (or earlier) SCN or time. This way, after starting Redo Apply, the physical standby databases will automatically begin applying redo data received from the primary database.

When flashing back primary or standby databases in this way, you do not have to be aware of past switchovers. Oracle can automatically flashback across past switchovers if the SCN/time is before any past switchover.

Note: Flashback Database must be enabled on the databases before the role transition occurs. See *Oracle Database Backup and Recovery User's Guide* for more information

---

8.4.1 Using Flashback Database After a Switchover

After a switchover, you can return databases to a time or system change number (SCN) prior to when the switchover occurred using the FLASHBACK DATABASE command.

If the switchover involved a physical standby database, the primary and standby database roles are preserved during the flashback operation. That is, the role in which the database is running does not change when the database is flashed back to the target SCN or time to which you flashed back the database. A database running in the

physical standby role after the switchover but prior to the flashback will still be running in the physical standby database role after the Flashback Database operation.

If the switchover involved a logical standby database, flashing back changes the role of the standby database to what it was at the target SCN or time to which you flashed back the database.

8.4.2 Using Flashback Database After a Failover

You can use Flashback Database to convert the failed primary database to a point in time before the failover occurred and then convert it into a standby database. See Section 13.2, "Converting a Failed Primary Into a Standby Database Using Flashback Database" for the complete step-by-step procedure.

9

Managing Physical and Snapshot Standby Databases

This chapter describes how to manage physical and snapshot standby databases. The following topics are discussed:

- Starting Up and Shutting Down a Physical Standby Database
- Opening a Physical Standby Database
- Primary Database Changes That Require Manual Intervention at a Physical Standby
- Recovering Through the OPEN RESETLOGS Statement
- Monitoring Primary, Physical Standby, and Snapshot Standby Databases
- Tuning Redo Apply
- Managing a Snapshot Standby Database

See *Oracle Data Guard Broker* to learn how the Data Guard broker simplifies the management of physical and snapshot standby databases.

9.1 Starting Up and Shutting Down a Physical Standby Database

This section describes how to start up and shut down a physical standby database.

9.1.1 Starting Up a Physical Standby Database

Use the SQL*Plus `STARTUP` command to start a physical standby database. The SQL*Plus STARTUP command starts, mounts, and opens a physical standby database in read-only mode when it is invoked without any arguments.

Once mounted or opened, a physical standby database can receive redo data from the primary database.

See Section 7.3 for information about Redo Apply and Section 9.2 for information about opening a physical standby database in read-only mode.

Note: When Redo Apply is started on a physical standby database that has not yet received redo data from the primary database, an `ORA-01112` message may be returned. This indicates that Redo Apply is unable to determine the starting sequence number for media recovery. If this occurs, manually retrieve an archived redo log file from the primary database and register it on the standby database, or wait for redo transport to begin before starting Redo Apply.

9.1.2 Shutting Down a Physical Standby Database

Use the SQL*Plus `SHUTDOWN` command to stop Redo Apply and shut down a physical standby database. 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 standby destination on the primary database and perform a log switch before shutting down the physical standby database.

9.2 Opening a Physical Standby Database

A physical standby database can be opened for read-only access and used to offload queries from a primary database.

If a license for the Oracle Active Data Guard option has been purchased, a physical standby database can be open while redo apply is active. This capability is known as real-time query. See Section 9.2.1 for more details.

If a license for the Oracle Active Data Guard option has not been purchased, a physical standby database cannot be open while redo apply is active, so the following rules must be observed when opening a physical standby database instance or starting redo apply:

- Redo apply must be stopped before any physical standby database instance is opened.
- If one or more physical standby instances are open, those instances must be closed before starting redo apply.

Note: The `SET TRANSACTION READ ONLY` SQL statement must be executed before performing a distributed query on a physical standby database.

9.2.1 Real-time query

A physical standby database can be open for read-only access while Redo Apply is active if a license for the Oracle Active Data Guard option has been purchased. This capability is known as real-time query.

A physical standby database instance cannot be opened if Redo Apply is active on that instance or on any other mounted instance. Use the following SQL statements to stop Redo Apply, open a standby instance read-only, and restart Redo Apply:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE CANCEL;
SQL> ALTER DATABASE OPEN;
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE USING CURRENT LOGFILE
  2> DISCONNECT;
```

Note: If Redo Apply is active on an open instance, any other physical standby database instance can be opened without having to stop apply.

Redo Apply cannot be started on a mounted physical standby instance if any other instance is open. The instance where Redo Apply is to be started must be opened before starting Redo Apply.

After a physical standby apply instance is terminated abnormally (for example, by a shutdown abort or a node crash), an attempt to open the physical standby database will result in an `ORA-16004: backup database requires recovery` error. If this happens, Redo Apply must be started and then stopped before you try again to open the physical standby database.

Example 9–1 illustrates the failure and recovery of a physical standby database instance named `Boston`, and then after recovery, opening `Boston` while Redo Apply is active (real-time query).

Example 9–1 Real-time query

At the start of this example, the physical standby instance `Boston` is mounted with Redo Apply active. Then `Boston` crashes due to a power outage and is re-started:

```
SQL> STARTUP
ORACLE instance started.

Total System Global Area  234364928 bytes
Fixed Size                  1298908 bytes
Variable Size             209718820 bytes
Database Buffers           16777216 bytes
Redo Buffers                6569984 bytes
Database mounted.
ORA-16004: backup database requires recovery
ORA-01196: file 1 is inconsistent due to a failed media recovery session
ORA-01110: data file 1: '/scratch/datafiles/oracle/dbs/system1.f'
```

The physical standby database is inconsistent because the `Boston` instance crashed while Redo Apply was active on it. Start Redo Apply so that the database can be recovered to a consistent SCN:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE DISCONNECT;
Database altered.
```

Wait for about a minute and then stop Redo Apply:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE CANCEL;
Database altered.
```

Note that the alert log will list the following warning which is normal and expected: `ORA-16037: user requested cancel of managed recovery operation`.

`Boston` is now recovered to a consistent SCN.

Open `Boston` and start Redo Apply:

```
SQL> ALTER DATABASE OPEN;
Database altered.

SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE USING
  2> CURRENT LOGFILE DISCONNECT;
```

```
Database altered.
```

Boston is now open and Redo Apply is active meaning that real-time query is in effect.

9.3 Primary Database Changes That Require Manual Intervention at a Physical Standby

Most structural changes made to a primary database are automatically propagated through redo data to a physical standby database. Table 9–1 lists primary database structural and configuration changes which require manual intervention at a physical standby database.

Table 9–1 Primary Database Changes That Require Manual Intervention at a Physical Standby

Reference	Primary Database Change	Action Required on Physical Standby Database
Section 9.3.1	Add a datafile or create a tablespace	No action is required if the STANDBY_FILE_MANAGEMENT database initialization parameter is set to AUTO. If this parameter is set to MANUAL, the new datafile must be copied to the physical standby database.
Section 9.3.2	Drop or delete a tablespace or datafile	Delete datafile from primary and physical standby database after the redo data containing the DROP or DELETE command is applied to the physical standby.
Section 9.3.3	Use transportable tablespaces	Move tablespace between the primary and the physical standby database.
Section 9.3.4	Rename a datafile	Rename the datafile on the physical standby database.
Section 9.3.5	Add or drop a redo log file group	Evaluate the configuration of the redo log and standby redo log on the physical standby database and adjust as necessary.
Section 9.3.6	Perform a DML or DDL operation using the NOLOGGING or UNRECOVERABLE clause	Copy the datafile containing the unlogged changes to the physical standby database.
Section 9.3.7	Grant or revoke administrative privileges or change the password of a user who has administrative privileges	If the REMOTE_LOGIN_PASSWORDFILE initialization parameter is set to SHARED or EXCLUSIVE, replace the password file on the physical standby database with a fresh copy of the password file from the primary database.
Section 9.3.8	Reset the TDE master encryption key	Replace the database encryption wallet on the physical standby database with a fresh copy of the database encryption wallet from the primary database.
Chapter 14	Change initialization parameters	Evaluate whether a corresponding change must be made to the initialization parameters on the physical standby database.

9.3.1 Adding a Datafile or Creating a Tablespace

The STANDBY_FILE_MANAGEMENT database initialization parameter controls whether the addition of a datafile to the primary database is automatically propagated to a physical standby databases.

- If the STANDBY_FILE_MANAGEMENT parameter on the physical standby database is set to AUTO, any new datafiles created on the primary database are automatically created on the physical standby database.
- If the STANDBY_FILE_MANAGEMENT database parameter on the physical standby database is set to MANUAL, a new datafile must be manually copied from the

primary database to the physical standby databases after it is added to the primary database.

Note that if an existing datafile from another database is copied to a primary database, that it must also be copied to the standby database and that the standby control file must be re-created, regardless of the setting of STANDBY_FILE_MANAGEMENT parameter.

9.3.1.1 Using the STANDBY_FILE_MANAGEMENT Parameter with Raw Devices

Note: Do not use the following procedure with databases that use Oracle Managed Files. Also, if the raw device path names are not the same on the primary and standby servers, use the DB_FILE_NAME_CONVERT database initialization parameter to convert the path names.

By setting the STANDBY_FILE_MANAGEMENT parameter to AUTO whenever new datafiles are added or dropped on the primary database, corresponding changes are made in the standby database without manual intervention. This is true as long as the standby database is using a file system. If the standby database is using raw devices for datafiles, then the STANDBY_FILE_MANAGEMENT parameter will continue to work, but manual intervention is needed. This manual intervention involves ensuring the raw devices exist before Redo Apply applies the redo data that will create the new datafile.

On the primary database, create a new tablespace where the datafiles reside in a raw device. At the same time, create the same raw device on the standby database. For example:

```
SQL> CREATE TABLESPACE MTS2 –
> DATAFILE '/dev/raw/raw100' size 1m;
Tablespace created.

SQL> ALTER SYSTEM SWITCH LOGFILE;
System altered.
```

The standby database automatically adds the datafile because the raw devices exist. The standby alert log shows the following:

```
Fri Apr  8 09:49:31 2005
Media Recovery Log /u01/MILLER/flash_recovery_area/MTS_STBY/archivelog/2005_04_
08/o1_mf_1_7_15ffgt0z_.arc
Recovery created file /dev/raw/raw100
Successfully added datafile 6 to media recovery
Datafile #6: '/dev/raw/raw100'
Media Recovery Waiting for thread 1 sequence 8 (in transit)
```

However, if the raw device was created on the primary system but not on the standby, then Redo Apply will stop due to file-creation errors. For example, issue the following statements on the primary database:

```
SQL> CREATE TABLESPACE MTS3 –
> DATAFILE '/dev/raw/raw101' size 1m;
Tablespace created.

SQL> ALTER SYSTEM SWITCH LOGFILE;
System altered.
```

The standby system does not have the /dev/raw/raw101 raw device created. The standby alert log shows the following messages when recovering the archive:

```
Fri Apr  8 10:00:22 2005
Media Recovery Log /u01/MILLER/flash_recovery_area/MTS_STBY/archivelog/2005_04_
08/o1_mf_1_8_15ffjrov_.arc
File #7 added to control file as 'UNNAMED00007'.
Originally created as:
'/dev/raw/raw101'
Recovery was unable to create the file as:
'/dev/raw/raw101'
MRP0: Background Media Recovery terminated with error 1274
Fri Apr  8 10:00:22 2005
Errors in file /u01/MILLER/MTS/dump/mts_mrp0_21851.trc:
ORA-01274: cannot add datafile '/dev/raw/raw101' - file could not be created
ORA-01119: error in creating database file '/dev/raw/raw101'
ORA-27041: unable to open file
Linux Error: 13: Permission denied
Additional information: 1
Some recovered datafiles maybe left media fuzzy
Media recovery may continue but open resetlogs may fail
Fri Apr  8 10:00:22 2005
Errors in file /u01/MILLER/MTS/dump/mts_mrp0_21851.trc:
ORA-01274: cannot add datafile '/dev/raw/raw101' - file could not be created
ORA-01119: error in creating database file '/dev/raw/raw101'
ORA-27041: unable to open file
Linux Error: 13: Permission denied
Additional information: 1
Fri Apr  8 10:00:22 2005
MTS; MRP0: Background Media Recovery process shutdown
ARCH: Connecting to console port...
```

9.3.1.2 Recovering from Errors

To correct the problems described in Section 9.3.1.1, perform the following steps:

1. Create the raw device on the standby database and assign permissions to the Oracle user.
2. Query the V$DATAFILE view. For example:

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

NAME.

--------------------------------------------------------------------------------
-
/u01/MILLER/MTS/system01.dbf
/u01/MILLER/MTS/undotbs01.dbf
/u01/MILLER/MTS/sysaux01.dbf
/u01/MILLER/MTS/users01.dbf
/u01/MILLER/MTS/mts.dbf
/dev/raw/raw100
/u01/app/oracle/product/10.1.0/dbs/UNNAMED00007

SQL> ALTER SYSTEM SET -
> STANDBY_FILE_MANAGEMENT=MANUAL;

SQL> ALTER DATABASE CREATE DATAFILE
2  '/u01/app/oracle/product/10.1.0/dbs/UNNAMED00007'
3  AS
4  '/dev/raw/raw101';
```

3. In the standby alert log you should see information similar to the following:

```
Fri Apr  8 10:09:30 2005
alter database create datafile
'/dev/raw/raw101' as '/dev/raw/raw101'

Fri Apr  8 10:09:30 2005
Completed: alter database create datafile
'/dev/raw/raw101' a
```

4. On the standby database, set STANDBY_FILE_MANAGEMENT to AUTO and restart Redo Apply:

```
SQL> ALTER SYSTEM SET STANDBY_FILE_MANAGEMENT=AUTO;
SQL> RECOVER MANAGED STANDBY DATABASE DISCONNECT;
```

At this point Redo Apply uses the new raw device datafile and recovery continues.

9.3.2 Dropping Tablespaces and Deleting Datafiles

When a tablespace is dropped or a datafile is deleted from a primary database, the corresponding datafile(s) must be deleted from the physical standby database. The following example shows how to drop a tablespace:

```
SQL> DROP TABLESPACE tbs_4;
SQL> ALTER SYSTEM SWITCH LOGFILE;
```

To verify that deleted datafiles are no longer part of the database, query the V$DATAFILE view.

Delete the corresponding datafile on the standby system after the redo data that contains the previous changes is applied to the standby database. For example:

```
% rm /disk1/oracle/oradata/payroll/s2tbs_4.dbf
```

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
```

9.3.2.1 Using DROP TABLESPACE INCLUDING CONTENTS AND DATAFILES

You can issue the SQL DROP TABLESPACE INCLUDING CONTENTS AND DATAFILES statement on the primary database to delete the datafiles on both the primary and standby databases. To use this statement, the STANDBY_FILE_MANAGEMENT initialization parameter must be set to AUTO. For example, to drop the tablespace at the primary site:

```
SQL> DROP TABLESPACE INCLUDING CONTENTS -
> AND DATAFILES tbs_4;
SQL> ALTER SYSTEM SWITCH LOGFILE;
```

9.3.3 Using Transportable Tablespaces with a Physical Standby Database

You can use the Oracle transportable tablespaces feature to move a subset of an Oracle database and plug it in to another Oracle database, essentially moving tablespaces between the databases.

To move or copy a set of tablespaces into a primary database when a physical standby is being used, perform the following steps:

1. Generate a transportable tablespace set that consists of datafiles for the set of tablespaces being transported and an export file containing structural information for the set of tablespaces.
2. Transport the tablespace set:
   a. Copy the datafiles and the export file to the primary database.
   b. Copy the datafiles to the standby database.

   The datafiles must be copied in a directory defined by the DB_FILE_NAME_CONVERT initialization parameter. If DB_FILE_NAME_CONVERT is *not* defined, then issue the ALTER DATABASE RENAME FILE statement to modify the standby control file *after* the redo data containing the transportable tablespace has been applied and has failed. The STANDBY_FILE_MANAGEMENT initialization parameter must be set to AUTO.
3. Plug in the tablespace.

   Invoke the Data Pump utility to plug the set of tablespaces into the primary database. Redo data will be generated and applied at the standby site to plug the tablespace into the standby database.

For more information about transportable tablespaces, see *Oracle Database Administrator's Guide.*

9.3.4 Renaming a Datafile in the Primary Database

When you rename one or more datafiles in the primary database, the change is not propagated to the standby database. Therefore, if you want to rename the same datafiles on the standby database, you must manually make the equivalent modifications on the standby database because the modifications are not performed automatically, even if the STANDBY_FILE_MANAGEMENT initialization parameter is set to AUTO.

The following steps describe how to rename a datafile in the primary database and manually propagate the changes to the standby database.

1. To rename the datafile in the primary database, take the tablespace offline:

```
SQL> ALTER TABLESPACE tbs_4 OFFLINE;
```

2. Exit from the SQL prompt and issue an operating system command, such as the following UNIX mv command, to rename the datafile on the primary system:

```
% mv /disk1/oracle/oradata/payroll/tbs_4.dbf
/disk1/oracle/oradata/payroll/tbs_x.dbf
```

3. Rename the datafile in the primary database and bring the tablespace back online:

```
SQL> ALTER TABLESPACE tbs_4 RENAME DATAFILE
  2> '/disk1/oracle/oradata/payroll/tbs_4.dbf'
  3>  TO '/disk1/oracle/oradata/payroll/tbs_x.dbf';
SQL> ALTER TABLESPACE tbs_4 ONLINE;
```

4. Connect to the standby database and stop Redo Apply:

```
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 USING CURRENT LOGFILE
  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'
```

9.3.5 Add or Drop a Redo Log File Group

The configuration of the redo log and standby redo log on a physical standby database should be reevaluated and adjusted as necessary after adding or dropping a redo log file group on the primary database.

Take the following steps to add or drop a redo log file group or standby redo log file group on a physical standby database:

1. Stop Redo Apply.
2. If the `STANDBY_FILE_MANAGEMENT` initialization parameter is set to `AUTO`, change the value to `MANUAL`.
3. Add or drop a log file group.
4. Restore the `STANDBY_FILE_MANAGEMENT` initialization parameter and the Redo Apply options to their original states.
5. Restart Redo Apply.

9.3.6 NOLOGGING or Unrecoverable Operations

When you perform a DML or DDL operation using the `NOLOGGING` or `UNRECOVERABLE` clause, the standby database is invalidated and may require substantial DBA administrative activities to repair. You can specify the `SQL ALTER DATABASE` or `SQL ALTER TABLESPACE` statement with the `FORCELOGGING` clause to override the `NOLOGGING` setting. However, this statement will not repair an already invalidated database.

See Section 13.4 for information about recovering after the `NOLOGGING` clause is used.

9.3.7 Refresh the Password File

If the REMOTE_LOGIN_PASSWORDFILE database initialization parameter is set to SHARED or EXCLUSIVE, the password file on a physical standby database must be replaced with a fresh copy from the primary database after granting or revoking administrative privileges or changing the password of a user with administrative privileges.

Failure to refresh the password file on the physical standby database may cause authentication of redo transport sessions or connections as SYSDBA or SYSOPER to the physical standby database to fail.

9.3.8 Reset the TDE Master Encryption Key

The database encryption wallet on a physical standby database must be replaced with a fresh copy of the database encryption wallet from the primary database whenever the TDE master encryption key is reset on the primary database.

Failure to refresh the database encryption wallet on the physical standby database will prevent access to encrypted columns on the physical standby database that are modified after the master encryption key is reset on the primary database.

9.4 Recovering Through the OPEN RESETLOGS Statement

Data Guard allows recovery on a physical standby database to continue after the primary database has been opened with the RESETLOGS option. When an ALTER DATABASE OPEN RESETLOGS statement is issued on the primary database, the incarnation of the database changes, creating a new branch of redo data.

When a physical standby database receives a new branch of redo data, Redo Apply automatically takes the new branch of redo data. For physical standby databases, no manual intervention is required if the standby database did not apply redo data past the new resetlogs SCN (past the start of the new branch of redo data). The following table describes how to resynchronize the standby database with the primary database branch.

If the standby database. . .	Then. . .	Perform these steps. . .
Has not applied redo data past the new resetlogs SCN (past the start of the new branch of redo data)	Redo Apply automatically takes the new branch of redo.	No manual intervention is necessary. The MRP automatically resynchronizes the standby database with the new branch of redo data.
Has applied redo data past the new resetlogs SCN (past the start of the new branch of redo data) and Flashback Database is enabled on the standby database	The standby database is recovered *in the future* of the new branch of redo data.	1. Follow the procedure in Section 13.3.1 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.
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.

If the standby database. . .	Then. . .	Perform these steps. . .
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 User's Guide* for more information about database incarnations, recovering through an `OPEN RESETLOGS` operation, and Flashback Database.

9.5 Monitoring Primary, Physical Standby, and Snapshot Standby Databases

This section describes where to find useful information for monitoring primary and standby databases.

Table 9–2 summarizes common primary database management actions and where to find information related to these actions.

Table 9–2 Sources of Information About Common Primary Database Management Actions

Primary Database Action	Primary Site Information	Standby Site Information
Enable or disable a redo thread	▪ Alert log ▪ `V$THREAD`	Alert log
Display database role, protection mode, protection level, switchover status, fast-start failover information, and so forth	`V$DATABASE`	`V$DATABASE`
Add or drop a redo log file group	▪ Alert log ▪ `V$LOG` ▪ `STATUS` column of `V$LOGFILE`	Alert log
`CREATE CONTROLFILE`	Alert log	Alert log
Monitor Redo Apply	▪ Alert log ▪ `V$ARCHIVE_DEST_STATUS`	▪ Alert log ▪ `V$ARCHIVED_LOG` ▪ `V$LOG_HISTORY` ▪ `V$MANAGED_STANDBY`
Change tablespace status	▪ `V$RECOVER_FILE` ▪ `DBA_TABLESPACES` ▪ Alert log	▪ `V$RECOVER_FILE` ▪ `DBA_TABLESPACES`
Add or drop a datafile or tablespace	▪ `DBA_DATA_FILES` ▪ Alert log	▪ `V$DATAFILE` ▪ Alert log
Rename a datafile	▪ `V$DATAFILE` ▪ Alert log	▪ `V$DATAFILE` ▪ Alert log
Unlogged or unrecoverable operations	▪ `V$DATAFILE` ▪ `V$DATABASE`	Alert log

Table 9–2 (Cont.) Sources of Information About Common Primary Database Management Actions

Primary Database Action	Primary Site Information	Standby Site Information
Monitor redo transport	■ V$ARCHIVE_DEST_STATUS ■ V$ARCHIVED_LOG ■ V$ARCHIVE_DEST ■ Alert log	■ V$ARCHIVED_LOG ■ Alert log
Issue OPEN RESETLOGS or CLEAR UNARCHIVED LOGFILES statements	Alert log	Alert log
Change initialization parameter	Alert log	Alert log

9.5.1 Using Views to Monitor Primary, Physical, and Snapshot Standby Databases

This section shows how to use dynamic performance views to monitor primary, physical standby, and snapshot standby databases.

The following dynamic performance views are discussed:

- V$DATABASE
- V$MANAGED_STANDBY
- V$ARCHIVED_LOG
- V$LOG_HISTORY
- V$DATAGUARD_STATUS

> **See Also:** *Oracle Database Reference* for complete reference information about views

9.5.1.1 V$DATABASE

The following query displays the data protection mode, data protection level, database role, and switchover status for a primary, physical standby or snapshot standby database:

```
SQL> SELECT PROTECTION_MODE, PROTECTION_LEVEL, -
> DATABASE_ROLE ROLE, SWITCHOVER_STATUS -
> FROM V$DATABASE;
```

The following query displays fast-start failover status:

```
SQL> SELECT FS_FAILOVER_STATUS "FSFO STATUS", -
> FS_FAILOVER_CURRENT_TARGET TARGET, -
> FS_FAILOVER_THRESHOLD THRESHOLD, -
> FS_FAILOVER_OBSERVER_PRESENT "OBSERVER PRESENT" -
> FROM V$DATABASE;
```

9.5.1.2 V$MANAGED_STANDBY

The following query displays Redo Apply and redo transport status on a physical standby database:

```
SQL> SELECT PROCESS, STATUS, THREAD#, SEQUENCE#,-
> BLOCK#, BLOCKS FROM V$MANAGED_STANDBY;

PROCESS STATUS       THREAD#    SEQUENCE#  BLOCK#     BLOCKS
------- ------------ ---------- ---------- ---------- ----------
RFS     ATTACHED     1          947        72         72
MRP0    APPLYING_LOG 1          946        10         72
```

The sample output shows that a RFS process completed archiving a redo log file with a sequence number of 947 and that Redo Apply is actively applying an archived redo log file with a sequence number of 946. Redo Apply is currently recovering block number 10 of the 72-block archived redo log file.

9.5.1.3 V$ARCHIVED_LOG

The following query displays information about archived redo log files that have been received by a physical or snapshot standby database from a primary database:

```
SQL> SELECT THREAD#, SEQUENCE#, FIRST_CHANGE#, -
> NEXT_CHANGE# FROM V$ARCHIVED_LOG;

THREAD#    SEQUENCE#  FIRST_CHANGE# NEXT_CHANGE#
---------- ---------- ------------- ------------
1          945        74651         74739
1          946        74739         74772
1          947        74772         7474
```

The sample output shows that three archived redo log files have been received from the primary database.

9.5.1.4 V$LOG_HISTORY

The following query displays archived log history information:

```
SQL> SELECT THREAD#, SEQUENCE#, FIRST_CHANGE#, -
> NEXT_CHANGE# FROM V$LOG_HISTORY;
```

9.5.1.5 V$DATAGUARD_STATUS

The following query displays messages generated by Data Guard events that caused a message to be written to the alert log or to a server process trace file:

```
SQL> SELECT MESSAGE FROM V$DATAGUARD_STATUS;
```

9.6 Tuning Redo Apply

The Oracle Data Guard Redo Apply and Media Recovery Best Practices white paper describes how to optimize Redo Apply and media recovery performance. This paper is available on the Oracle Maximum Availability Architecture (MAA) home page at:

http://otn.oracle.com/deploy/availability/htdocs/maa.htm

> **See Also:** Oracle MetaLink note 454848.1 at
>
> https://metalink.oracle.com for information about the installation and use of the Standby Statspack, which can be used to collect redo apply performance data from a physical standby database

9.7 Managing a Snapshot Standby Database

A snapshot standby database is a fully updatable standby database that is created by converting a physical standby database into a snapshot standby database. A snapshot standby database receives and archives, but does not apply, redo data from a primary database. Redo data received from the primary database is applied when a snapshot

standby database is converted back into a physical standby database, after discarding all local updates to the snapshot standby database.

A snapshot standby database typically diverges from its primary database over time because redo data from the primary database is not applied as it is received. Local updates to the snapshot standby database will cause additional divergence. The data in the primary database is fully protected however, because a snapshot standby can be converted back into a physical standby database at any time, and the redo data received from the primary will then be applied.

A snapshot standby database provides disaster recovery and data protection benefits that are similar to those of a physical standby database. Snapshot standby databases are best used in scenarios where the benefit of having a temporary, updatable snapshot of the primary database justifies additional administrative complexity and increased time to recover from primary database failures.

9.7.1 Converting a Physical Standby Database into a Snapshot Standby Database

Perform the following steps to convert a physical standby database into a snapshot standby database:

1. Stop Redo Apply, if it is active.
2. On an Oracle Real Applications Cluster (RAC) database, shut down all but one instance.
3. Ensure that the database is mounted, but not open.
4. Issue the following SQL statement to perform the conversion:

```
SQL> ALTER DATABASE CONVERT TO SNAPSHOT STANDBY;
```

The database is dismounted after conversion and must be restarted.

> **Note:** A physical standby database that is managed by the Data Guard broker can be converted into a snapshot standby database using either DGMGRL or Oracle Enterprise Manager. See *Oracle Data Guard Broker* for more details.

9.7.2 Using a Snapshot Standby Database

A snapshot standby database can be opened in read-write mode and is fully updatable.

A snapshot standby database has the following characteristics:

- A snapshot standby database cannot be the target of a switchover or failover. A snapshot standby database must first be converted back into a physical standby database before performing a role transition to it.
- A snapshot standby database cannot be the only standby database in a Maximum Protection Data Guard configuration.

> **Note:** Flashback Database is used to convert a snapshot standby database back into a physical standby database. Any operation that cannot be reversed using Flashback Database technology will prevent a snapshot standby from being converted back to a physical standby.

9.7.3 Converting a Snapshot Standby Database into a Physical Standby Database

Perform the following steps to convert a snapshot standby database into a physical standby database:

1. On an Oracle Real Applications Cluster (RAC) database, shut down all but one instance.
2. Ensure that the database is mounted, but not open.
3. Issue the following SQL statement to perform the conversion:

```
SQL> ALTER DATABASE CONVERT TO PHYSICAL STANDBY;
```

The database is dismounted after conversion and must be restarted.

Redo data received while the database was a snapshot standby database will be automatically applied when Redo Apply is started.

Note: A snapshot standby database must be opened at least once in read-write mode before it can be converted into a physical standby database.

10

Managing a Logical Standby Database

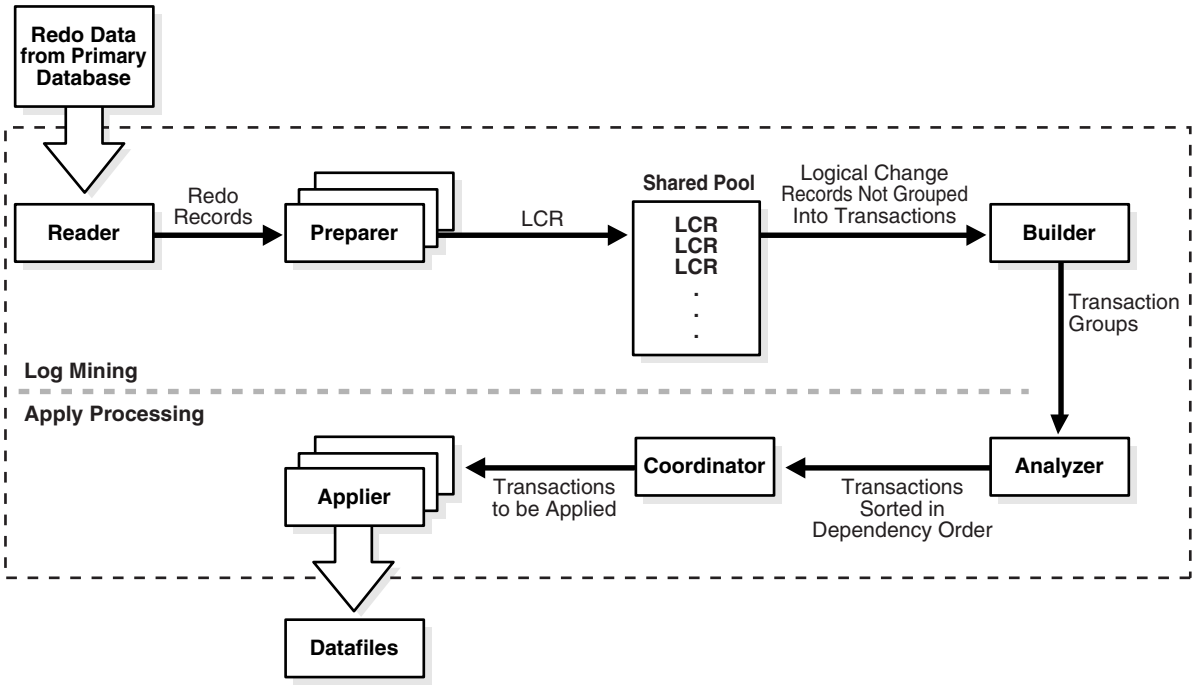
This chapter contains the following topics:

10.1 Overview of the SQL Apply Architecture

SQL Apply uses a collection of background processes to apply changes from the primary database to the logical standby database.

Figure 10–1 shows the flow of information and the role that each process performs.

Figure 10–1 SQL Apply Processing



The different processes involved and their functions during log mining and apply processing are as follows:

During log mining:

- The `READER` process reads redo records from the archived redo log files or standby redo log files.
- The `PREPARER` process converts block changes contained in redo records into logical change records (LCRs). Multiple `PREPARER` processes can be active for a given redo log file. The LCRs are staged in the system global area (SGA), known as the *LCR cache*.
- The `BUILDER` process groups LCRs into transactions, and performs other tasks, such as memory management in the LCR cache, checkpointing related to SQL Apply restart and filtering out of uninteresting changes.

During apply processing:

- The `ANALYZER` process identifies dependencies between different transactions.
- The `COORDINATOR` process (LSP) assigns transactions to different appliers and coordinates among them to ensure that dependencies between transactions are honored.
- The `APPLIER` processes applies transactions to the logical standby database under the supervision of the coordinator process.

You can query the `V$LOGSTDBY_PROCESS` view to examine the activity of the SQL Apply processes. Another view that provides information about current activity is the `V$LOGSTDBY_STATS` view that displays statistics, current state, and status information for the logical standby database during SQL Apply activities. These and other relevant views are discussed in more detail in Section 10.3, "Views Related to Managing and Monitoring a Logical Standby Database".

Note: All SQL Apply processes (including the coordinator process `lsp0`) are true background processes. They are not regulated by resource manager. Therefore, creating resource groups at the logical standby database does not affect the SQL Apply processes.

10.1.1 Various Considerations for SQL Apply

This section contains the following topics:

- Transaction Size Considerations
- Pageout Considerations
- Restart Considerations
- DML Apply Considerations
- DDL Apply Considerations
- Password Verification Functions

10.1.1.1 Transaction Size Considerations

SQL Apply categorizes transactions into two classes: small and large:

- Small transactions—SQL Apply starts applying LCRs belonging to a small transaction once it has encountered the commit record for the transaction in the redo log files.
- Large transactions—SQL Apply breaks large transactions into smaller pieces called *transaction chunks*, and starts applying the chunks before the commit record for the large transaction is seen in the redo log files. This is done to reduce memory pressure on the LCR cache and to reduce the overall failover time.

  For example, without breaking into smaller pieces, a SQL*Loader load of ten million rows, each 100 bytes in size, would use more than 1 GB of memory in the LCR cache. If the memory allocated to the LCR cache was less than 1 GB, it would result in pageouts from the LCR cache.

  Apart from the memory considerations, if SQL Apply did not start applying the changes related to the ten million row SQL*Loader load until it encountered the `COMMIT` record for the transaction, it could stall a role transition. A switchover or a failover that is initiated after the transaction commit cannot finish until SQL Apply has applied the transaction on the logical standby database.

  Despite the use of transaction chunks, SQL Apply performance may degrade when processing transactions that modify more than one million rows. Oracle recommends that the size of transactions be limited to less than a million rows. For a large SQL*Loader table load, use the `ROWS` clause to limit the number of rows loaded within a transaction.

All transactions start out categorized as small transactions. Depending on the amount of memory available for the LCR cache and the amount of memory consumed by LCRs belonging to a transaction, SQL Apply determines when to recategorize a transaction as a large transaction.

10.1.1.2 Pageout Considerations

Pageouts occur in the context of SQL Apply when memory in the LCR cache is exhausted and space needs to be released for SQL Apply to make progress.

For example, assume the memory allocated to the LCR cache is 100 MB and SQL Apply encounters an INSERT transaction to a table with a LONG column of size 300 MB. In this case, the log-mining component will page out the first part of the LONG data to read the later part of the column modification. In a well-tuned logical standby database, pageout activities will occur occasionally and should not effect the overall throughput of the system.

> **See Also:** See Section 10.5, "Customizing a Logical Standby Database" for more information about how to identify problematic pageouts and perform corrective actions

10.1.1.3 Restart Considerations

Modifications made to the logical standby database do not become persistent until the commit record of the transaction is mined from the redo log files and applied to the logical standby database. Thus, every time SQL Apply is stopped, whether as a result of a user directive or because of a system failure, SQL Apply must go back and mine the earliest uncommitted transaction again.

In cases where a transaction does little work but remains open for a long period of time, restarting SQL Apply from the start could be prohibitively costly because SQL Apply would have to mine a large number of archived redo log files again, just to read the redo data for a few uncommitted transactions. To mitigate this, SQL Apply periodically checkpoints old uncommitted data. The SCN at which the checkpoint is taken is reflected in the RESTART_SCN column of V$LOGSTDBY_PROGRESS view.

Upon restarting, SQL Apply starts mining redo records that are generated at an SCN greater than value shown by the RESTART_SCN column. Archived redo log files that are not needed for restart are automatically deleted by SQL Apply.

Certain workloads, such as large DDL transactions, parallel DML statements (PDML), and direct-path loads, will prevent the RESTART_SCN from advancing for the duration of the workload.

10.1.1.4 DML Apply Considerations

SQL Apply has the following characteristics when applying DML transactions that affect the throughput and latency on the logical standby database:

- Batch updates or deletes done on the primary database, where a single statement results in multiple rows being modified, are applied as individual row modifications on the logical standby database. Thus, it is imperative for each maintained table to have a unique index or a primary key. See Section 4.1.2, "Ensure Table Rows in the Primary Database Can Be Uniquely Identified" for more information.
- Direct path inserts performed on the primary database are applied using a conventional INSERT statement on the logical standby database.
- Parallel DML (PDML) transactions are not executed in parallel on the logical standby database.

10.1.1.5 DDL Apply Considerations

SQL Apply has the following characteristics when applying DDL transactions that affect the throughput and latency on the logical standby database:

- DDL transactions are applied serially on the logical standby database. Thus, DDL transactions applied concurrently on the primary database are applied one at a time on the logical standby database.

- CREATE TABLE AS SELECT (CTAS) statements are executed such that the DML activities (that are part of the CTAS statement) are suppressed on the logical standby database. The rows inserted in the newly created table as part of the CTAS statement are mined from the redo log files and applied to the logical standby database using INSERT statements.
- SQL Apply reissues the DDL that was performed at the primary database, and ensures that DMLs that occur within the same transaction on the same object that is the target of the DDL operation are not replicated at the logical standby database. Thus, the following two cases will cause the primary and standby sites to diverge from each other:
  - The DDL contains a non-literal value that is derived from the state at the primary database. An example of such a DDL is:

    ```
    ALTER TABLE hr.employees ADD (start_date date default sysdate);
    ```

    Because SQL Apply will reissue the same DDL at the logical standby, the function sysdate() will be reevaluated at the logical standby. Thus, the column start_date will be created with a different default value than at the primary database.
  - The DDL fires DML triggers defined on the target table. Since the triggered DMLs occur in the same transaction as the DDL, and operate on the table that is the target of the DDL, these triggered DMLs will not be replicated at the logical standby.

    For example, assume you create a table as follows:

    ```
     create table HR.TEMP_EMPLOYEES (
     emp_id       number primary key,
     first_name   varchar2(64),
     last_name    varchar2(64),
     modify_date  timestamp);
    ```

    Assume you then create a trigger on the table such that any time the table is updated the modify_date is updated to reflect the time of change:

    ```
     CREATE OR REPLACE TRIGGER TRG_TEST_MOD_DT  BEFORE UPDATE ON HR.TEST_
    EMPLOYEES
     REFERENCING
     NEW  AS NEW_ROW  FOR EACH ROW
     BEGIN
     :NEW_ROW.MODIFY_DATE:= SYSTIMESTAMP;
     END;
    /
    ```

    This table will be maintained correctly under the usual DML/DDL workload. However if you add a column with the default value to the table, the ADD COLUMN DDL fires this update trigger and changes the MODIFY_DATE column of all rows in the table to a new timestamp. These changes to the MODIFY_DATE column are not replicated at the logical standby database. Subsequent DMLs to the table will stop SQL Apply because the MODIFY_DATE column data recorded in the redo stream will not match the data that exists at the logical standby database.

10.1.1.6 Password Verification Functions

Password verification functions that check for the complexity of passwords must be created in the SYS schema. Because SQL Apply does not replicate objects created in the SYS schema, such verification functions will not be replicated to the logical standby

database. You must create the password verification function manually at the logical standby database, and associate it with the appropriate profiles.

10.2 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 Oracle9i. 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 10.5.4.

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

10.3 Views Related to Managing and Monitoring a Logical Standby Database

The following performance views monitor the behavior of SQL Apply maintaining a logical standby database. The following sections describe the key views that can be used to monitor a logical standby database:

- DBA_LOGSTDBY_EVENTS View
- DBA_LOGSTDBY_LOG View
- V$DATAGUARD_STATS View
- V$LOGSTDBY_PROCESS View
- V$LOGSTDBY_PROGRESS View
- V$LOGSTDBY_STATE View
- V$LOGSTDBY_STATS View

See Also: *Oracle Database Reference* for complete reference information about views

10.3.1 DBA_LOGSTDBY_EVENTS View

The DBA_LOGSTDBY_EVENTS view records interesting events that occurred during the operation of SQL Apply. By default, the view records the most recent 10,000 events. However, you can change the number of recorded events by calling DBMS_LOGSTDBY.APPLY_SET() PL/SQL procedure. If SQL Apply should stop unexpectedly, the reason for the problem is also recorded in this view.

Note: Errors that cause SQL Apply to stop are recorded in the events table These events are put into the ALERT.LOG file as well, with the LOGSTDBY keyword included in the text. When querying the view, select the columns in order by EVENT_TIME_STAMP, COMMIT_SCN, and CURRENT_SCN to ensure the desired ordering of events.

The view can be customized to contain other information, such as which DDL transactions were applied and which were skipped. For example:

```
SQL> ALTER SESSION SET NLS_DATE_FORMAT  = 'DD-MON-YY HH24:MI:SS';
Session altered.
SQL> COLUMN STATUS FORMAT A60
SQL> SELECT EVENT_TIME, STATUS, EVENT FROM DBA_LOGSTDBY_EVENTS
  2  ORDER BY EVENT_TIMESTAMP, COMMIT_SCN, CURRENT_SCN;

EVENT_TIME         STATUS
------------------------------------------------------------------------------
EVENT
------------------------------------------------------------------------------
23-JUL-02 18:20:12 ORA-16111: log mining and apply setting up
23-JUL-02 18:25:12 ORA-16128: User initiated shut down successfully completed
23-JUL-02 18:27:12 ORA-16112: log mining and apply stopping
23-JUL-02 18:55:12 ORA-16128: User initiated shut down successfully completed
23-JUL-02 18:57:09 ORA-16111: log mining and apply setting up
23-JUL-02 20:21:47 ORA-16204: DDL successfully applied
create table hr.test_emp (empno number, ename varchar2(64))
23-JUL-02 20:22:55 ORA-16205: DDL skipped due to skip setting
create database link link_to_boston connect to system identified by change_on_inst
7 rows selected.
```

This query shows that SQL Apply was started and stopped a few times. It also shows what DDL was applied and skipped.

10.3.2 DBA_LOGSTDBY_LOG View

The DBA_LOGSTDBY_LOG view provides dynamic information about archived logs being processed by SQL Apply.

For example:

```
SQL> COLUMN DICT_BEGIN FORMAT A10;
SQL> SET NUMF 99999999
SQL> SELECT FILE_NAME, SEQUENCE# AS SEQ#, FIRST_CHANGE# AS F_SCN#, -
     NEXT_CHANGE# AS N_SCN#, TIMESTAMP, -
     DICT_BEGIN AS BEG, DICT_END AS END, -
     THREAD# AS THR#, APPLIED FROM DBA_LOGSTDBY_LOG -
     ORDER BY SEQUENCE#;
```

```
FILE_NAME                 SEQ# F_SCN    N_SCN TIMESTAM BEG END THR# APPLIED
------------------------- ---- -------- -------- -------- --- --- --- ---------
/oracle/dbs/hq_nyc_2.log  2     101579  101588 11:02:58 NO  NO  1     YES
/oracle/dbs/hq_nyc_3.log  3     101588  142065 11:02:02 NO  NO  1     YES
/oracle/dbs/hq_nyc_4.log  4     142065  142307 11:02:10 NO  NO  1     YES
/oracle/dbs/hq_nyc_5.log  5     142307  142739 11:02:48 YES YES 1     YES
/oracle/dbs/hq_nyc_6.log  6     142739  143973 12:02:10 NO  NO  1     YES
/oracle/dbs/hq_nyc_7.log  7     143973  144042 01:02:11 NO  NO  1     YES
/oracle/dbs/hq_nyc_8.log  8     144042  144051 01:02:01 NO  NO  1     YES
/oracle/dbs/hq_nyc_9.log  9     144051  144054 01:02:16 NO  NO  1     YES
/oracle/dbs/hq_nyc_10.log 10    144054  144057 01:02:21 NO  NO  1     YES
/oracle/dbs/hq_nyc_11.log 11    144057  144060 01:02:26 NO  NO  1  CURRENT
/oracle/dbs/hq_nyc_12.log 12    144060  144089 01:02:30 NO  NO  1  CURRENT
/oracle/dbs/hq_nyc_13.log 13    144089  144147 01:02:41 NO  NO  1       NO
```

The `YES` entries in the `BEG` and `END` columns indicate that a LogMiner dictionary build starts at log file sequence number 5. The most recent archived redo log file is sequence number 13, and it was received at the logical standby database at 01:02:41.

The `APPLIED` column indicates that SQL Apply has applied all redo before SCN 144057. Since transactions can span multiple archived log files, multiple archived log files may show the value `CURRENT` in the `APPLIED` column.

10.3.3 V$DATAGUARD_STATS View

This view provides information related to the failover characteristics of the logical standby database, including:

- The time to failover (`apply finish time`)
- How current is the committed data in the logical standby database (`apply lag`)
- What the potential data loss will be in the event of a disaster (`transport lag`).

For example:

```
SQL> COL NAME FORMAT A20
SQL> COL VALUE FORMAT A12
SQL> COL UNIT FORMAT A30
SQL> SELECT NAME, VALUE, UNIT FROM V$DATAGUARD_STATS;

NAME                 VALUE        UNIT
-------------------- ------------ ------------------------------
apply finish time    +00 00:00:00 day(2) to second(1) interval
apply lag            +00 00:00:00 day(2) to second(0) interval
transport lag        +00 00:00:00 day(2) to second(0) interval
```

This output is from a logical standby database that has received and applied all redo generated from the primary database.

10.3.4 V$LOGSTDBY_PROCESS View

This view provides information about the current state of the various processes involved with SQL Apply, including;

- Identifying information (`sid | serial# | spid`)
- SQL Apply process: `COORDINATOR`, `READER`, `BUILDER`, `PREPARER`, `ANALYZER`, or `APPLIER` (`type`)
- Status of the process's current activity (`status_code | status`)

- Highest redo record processed by this process (`high_scn`)

For example:

```
SQL> COLUMN SERIAL# FORMAT 9999
SQL> COLUMN SID FORMAT 9999
SQL> SELECT SID, SERIAL#, SPID, TYPE, HIGH_SCN FROM V$LOGSTDBY_PROCESS;

  SID   SERIAL#   SPID         TYPE            HIGH_SCN
 -----  -------   -----------  ---------------- ----------
   48        6    11074        COORDINATOR     7178242899
   56       56    10858        READER          7178243497
   46        1    10860        BUILDER         7178242901
   45        1    10862        PREPARER        7178243295
   37        1    10864        ANALYZER        7178242900
   36        1    10866        APPLIER         7178239467
   35        3    10868        APPLIER         7178239463
   34        7    10870        APPLIER         7178239461
   33        1    10872        APPLIER         7178239472

9 rows selected.
```

The `HIGH_SCN` column shows that the reader process is ahead of all other processes, and the `PREPARER` and `BUILDER` process ahead of the rest.

```
SQL> COLUMN STATUS FORMAT A40
SQL> SELECT TYPE, STATUS_CODE, STATUS FROM V$LOGSTDBY_PROCESS;

TYPE             STATUS_CODE STATUS
---------------- ----------- ----------------------------------------
COORDINATOR            16117 ORA-16117: processing
READER                 16127 ORA-16127: stalled waiting for additional
                             transactions to be applied
BUILDER                16116 ORA-16116: no work available
PREPARER               16116 ORA-16117: processing
ANALYZER               16120 ORA-16120: dependencies being computed for
                             transaction at SCN 0x0001.abdb440a
APPLIER                16124 ORA-16124: transaction 1 13 1427 is waiting
                             on another transaction
APPLIER                16121 ORA-16121: applying transaction with commit
                             SCN 0x0001.abdb4390
APPLIER                16123 ORA-16123: transaction 1 23  1231 is waiting
                             for commit approval
APPLIER                16116 ORA-16116: no work available
```

The output shows a snapshot of SQL Apply running. On the mining side, the `READER` process is waiting for additional memory to become available before it can read more, the `PREPARER` process is processing redo records, and the `BUILDER` process has no work available. On the apply side, the `COORDINATOR` is assigning more transactions to `APPLIER` processes, the `ANALYZER` is computing dependencies at SCN 7178241034, one `APPLIER` has no work available, while two have outstanding dependencies that are not yet satisfied.

> **See Also:** Section 10.4.1, "Monitoring SQL Apply Progress" for example output

10.3.5 V$LOGSTDBY_PROGRESS View

This view provides detailed information regarding progress made by SQL Apply, including:

- SCN and time at which all transactions that have been committed on the primary database have been applied to the logical standby database (`applied_scn`, `applied_time`)
- SCN and time at which SQL Apply would begin reading redo records (`restart_scn`, `restart_time`) on restart
- SCN and time of the latest redo record received on the logical standby database (`latest_scn`, `latest_time`)
- SCN and time of the latest record processed by the `BUILDER` process (`mining_scn`, `mining_time`)

For example:

```
SQL> SELECT APPLIED_SCN, LATEST_SCN, MINING_SCN, RESTART_SCN FROM V$LOGSTDBY_
PROGRESS;

APPLIED_SCN  LATEST_SCN MINING_SCN RESTART_SCN
----------- ----------- ---------- -----------
 7178240496  7178240507 7178240507  7178219805
```

According to the output:

- SQL Apply has applied all transactions committed on or before SCN of 7178240496
- The latest redo record received at the logical standby database was generated at SCN 7178240507
- The mining component has processed all redo records generate on or before SCN 7178240507
- If SQL Apply stops and restarts for any reason, it will start mining redo records generated on or after SCN 7178219805

```
SQL> ALTER SESSION SET NLS_DATE_FORMAT='yy-mm-dd hh24:mi:ss';
Session altered

SQL> SELECT APPLIED_TIME, LATEST_TIME, MINING_TIME, RESTART_TIME FROM V$LOGSTDBY_
PROGRESS;

APPLIED_TIME      LATEST_TIME       MINING_TIME       RESTART_TIME
----------------- ----------------- ----------------- -----------------
05-05-12 10:38:21 05-05-12 10:41:53 05-05-12 10:41:21 05-05-12 10:09:30
```

According to the output:

- SQL Apply has applied all transactions committed on or before the time 05-05-12 10:38:21 (`APPLIED_TIME`)
- The last redo was generated at time 05-05-12 10:41:53 at the primary database (`LATEST_TIME`)
- The mining engine has processed all redo records generated on or before 05-05-12 10:41:21 (`MINING_TIME`)
- In the event of a restart, SQL Apply will start mining redo records generated after the time 05-05-12 10:09:30

> **See Also:** Section 10.4.1, "Monitoring SQL Apply Progress" for example output

10.3.6 V$LOGSTDBY_STATE View

This view provides a synopsis of the current state of SQL Apply, including:

- The DBID of the primary database (`primary_dbid`).
- The LogMiner session ID allocated to SQL Apply (`session_id`).
- Whether or not SQL Apply is applying in real time (`realtime_apply`).

For example:

```
SQL> COLUMN REALTIME_APPLY FORMAT a15
SQL> COLUMN STATE FORMAT a16
SQL> SELECT * FROM V$LOGSTDBY_STATE;

PRIMARY_DBID SESSION_ID REALTIME_APPLY  STATE
------------ ---------- --------------- ----------------
  1562626987          1 Y               APPLYING
```

The output shows that SQL Apply is running in the real-time apply mode and is currently applying redo data received from the primary database, the primary database's `DBID` is 1562626987 and the LogMiner session identifier associated the SQL Apply session is 1.

> **See Also:** Section 10.4.1, "Monitoring SQL Apply Progress" for example output

10.3.7 V$LOGSTDBY_STATS View

The `V$LOGSTDBY_STATS` view displays statistics, current state, and status information related to SQL apply. No rows are returned from this view when SQL Apply is not running. This view is only meaningful in the context of a logical standby database.

For example:

```
SQL> ALTER SESSION SET_NLS_DATE_FORMAT='dd-mm-yyyy hh24:mi:ss';
Session altered

SQL> SELECT SUBSTR(name, 1, 40) AS NAME, SUBSTR(value,1,32) AS VALUE FROM V$LOGSTDBY_STATS;

NAME                                     VALUE
---------------------------------------- --------------------------------
logminer session id                      1
number of preparers                      1
number of appliers                       5
server processes in use                  9
maximum SGA for LCR cache (MB)           30
maximum events recorded                  10000
preserve commit order                    TRUE
transaction consistency                  FULL
record skipped errors                    Y
record skipped DDLs                      Y
record applied DDLs                      N
record unsupported operations            N
realtime apply                           Y
apply delay (minutes)                    0
coordinator state                        APPLYING
coordinator startup time                 19-06-2007 09:55:47
coordinator uptime (seconds)             3593
txns received from logminer              56
```

```
txns assigned to apply                  23
txns applied                            22
txns discarded during restart           33
large txns waiting to be assigned       2
rolled back txns mined                  4
DDL txns mined                          40
CTAS txns mined                         0
bytes of redo mined                     60164040
bytes paged out                         0
pageout time (seconds)                  0
bytes checkpointed                      4845
checkpoint time (seconds)               0
system idle time (seconds)              2921
standby redo logs mined                 0
archived logs mined                     5
gap fetched logs mined                  0
standby redo log reuse detected         1
logfile open failures                   0
current logfile wait (seconds)          0
total logfile wait (seconds)            2910
thread enable mined                     0
thread disable mined                    0
.
40 rows selected.
```

10.4 Monitoring a Logical Standby Database

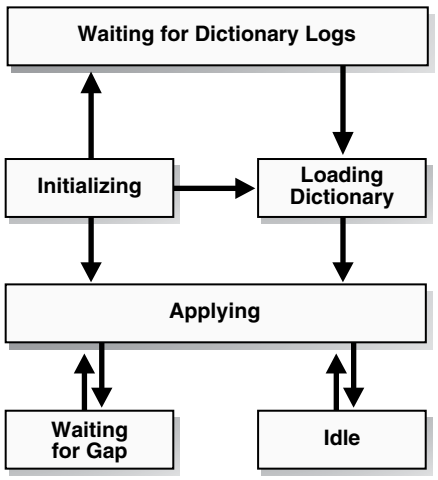
This section contains the following topics:

- Monitoring SQL Apply Progress
- Automatic Deletion of Log Files

10.4.1 Monitoring SQL Apply Progress

SQL Apply can be in any of six states of progress: initializing SQL Apply, waiting for dictionary logs, loading the LogMiner dictionary, applying (redo data), waiting for an archive gap to be resolved, and idle. Figure 10–2 shows the flow of these states.

Figure 10–2 Progress States During SQL Apply Processing



The following subsections describe each state in more detail.

Initializing State

When you start SQL Apply by issuing `ALTER DATABASE START LOGICAL STANDBY APPLY` statement, it goes in the *initializing* state.

To determine the current state of SQL Apply, query the `V$LOGSTDBY_STATE` view. For example:

```
SQL> SELECT SESSION_ID, STATE FROM V$LOGSTDBY_STATE;

SESSION_ID    STATE
----------    -------------
1             INITIALIZING
```

The `SESSION_ID` column identifies the persistent LogMiner session created by SQL Apply to mine the archived redo log files generated by the primary database.

Waiting for Dictionary Logs

The first time the SQL Apply is started, it needs to load the LogMiner dictionary captured in the redo log files. SQL Apply will stay in the `WAITING FOR DICTIONARY LOGS` state until it has received all redo data required to load the LogMiner dictionary.

Loading Dictionary State

This *loading dictionary* state can persist for a while. Loading the LogMiner dictionary on a large database can take a long time. Querying the `V$LOGSTDBY_STATE` view returns the following output when loading the dictionary:

```
SQL> SELECT SESSION_ID, STATE FROM V$LOGSTDBY_STATE;

SESSION_ID    STATE
----------    ------------------
1             LOADING DICTIONARY
```

Only the `COORDINATOR` process and the mining processes are spawned until the LogMiner dictionary is fully loaded. Therefore, if you query the `V$LOGSTDBY_PROCESS` at this point, you will not see any of the `APPLIER` processes. For example:

```
SQL> SELECT SID, SERIAL#, SPID, TYPE FROM V$LOGSTDBY_PROCESS;

SID     SERIAL#     SPID       TYPE
------  ---------   ---------  --------------------
47      3           11438      COORDINATOR
50      7           11334      READER
45      1           11336      BUILDER
44      2           11338      PREPARER
43      2           11340      PREPARER
```

You can get more detailed information about the progress in loading the dictionary by querying the `V$LOGMNR_DICTIONARY_LOAD` view. The dictionary load happens in three phases:

1. The relevant archived redo log files or standby redo logs files are mined to gather the redo changes relevant to load the LogMiner dictionary.
2. The changes are processed and loaded in staging tables inside the database.
3. The LogMiner dictionary tables are loaded by issuing a series of DDL statements.

For example:

```
SQL> SELECT PERCENT_DONE, COMMAND
     FROM   V$LOGMNR_DICTIONARY_LOAD
     WHERE  SESSION_ID = (SELECT SESSION_ID FROM V$LOGSTDBY_STATE);

PERCENT_DONE     COMMAND
-------------    ------------------------------
40               alter table SYSTEM.LOGMNR_CCOL$ exchange partition
                 P101 with table SYS.LOGMNRLT_101_CCOL$ excluding
                 indexes without validation
```

If the PERCENT_DONE or the COMMAND column does not change for a long time, query the V$SESSION_LONGOPS view to monitor the progress of the DDL transaction in question.

Applying State

In this state, SQL Apply has successfully loaded the initial snapshot of the LogMiner dictionary, and is currently applying redo data to the logical standby database.

For detailed information about the SQL Apply progress, query the V$LOGSTDBY_PROGRESS view:

```
SQL> ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-YYYY HH24:MI:SS';
SQL> SELECT APPLIED_TIME, APPLIED_SCN, MINING_TIME, MINING_SCN,
     FROM V$LOGSTDBY_PROGRESS;

APPLIED_TIME            APPLIED_SCN   MINING_TIME           MINING_SCN
--------------------    -----------   --------------------  -----------
10-JAN-2005 12:00:05    346791023     10-JAN-2005 12:10:05  3468810134
```

All committed transactions seen at or before APPLIED_SCN (or APPLIED_TIME) on the primary database have been applied to the logical standby database. The mining engine has processed all redo records generated at or before MINING_SCN (and MINING_TIME) on the primary database. At steady state, the value of MINING_SCN (and MINING_TIME) will always be ahead of APPLIED_SCN (and APPLIED_TIME).

Waiting On Gap State

This state occurs when SQL Apply has mined and applied all available redo records, and is waiting for a new log file (or a missing log file) to be archived by the RFS process.

```
SQL> SELECT STATUS FROM V$LOGSTDBY_PROCESS WHERE TYPE = 'READER';

STATUS
------------------------------------------------------------------------
ORA:01291 Waiting for logfile
```

Idle State

SQL Apply enters this state once it has applied all redo generated by the primary database.

10.4.2 Automatic Deletion of Log Files

Foreign archived logs contain redo that was shipped from the primary database. There are two ways to store foreign archive logs:

- In the flash recovery area
- In a directory outside of the flash recovery area

Foreign archived logs stored in the flash recovery area are always managed by SQL Apply. After all redo records contained in the log have been applied at the logical standby database, they are retained for the time period specified by the DB_FLASHBACK_RETENTION_TARGET parameter (or for 1440 minutes if DB_FLASHBACK_RETENTION_TARGET is not specified). You cannot override automatic management of foreign archived logs that are stored in the flash recovery area.

Foreign archived logs that are not stored in flash recovery area are by default managed by SQL Apply. Under automatic management, foreign archived logs that are not stored in the flash recovery area are retained for the time period specified by the LOG_AUTO_DEL_RETENTION_TARGET parameter once all redo records contained in the log have been applied at the logical standby database. You can override automatic management of foreign archived logs not stored in flash recovery area by executing the following PL/SQL procedure:

```
SQL> EXECUTE DBMS_LOGSTDBY.APPLY_SET('LOG_AUTO_DELETE', FALSE);
```

> **Note:** Use the DBMS_LOGTSDBY.APPLY_SET procedure to set this parameter. If you do not specify LOG_AUTO_DEL_RETENTION_TARGET explicitly, it defaults to DB_FLASHBACK_RETENTION_TARGET set in the logical standby database, or to 1440 minutes in case DB_FLASHBACK_RETENTION_TARGET is not set.

If you are overriding the default automatic log deletion capability, periodically perform the following steps to identify and delete archived redo log files that are no longer needed by SQL Apply:

1. To purge the logical standby session of metadata that is no longer needed, enter the following PL/SQL statement:

   ```
   SQL> EXECUTE DBMS_LOGSTDBY.PURGE_SESSION;
   ```

   This statement also updates the DBA_LOGMNR_PURGED_LOG view that displays the archived redo log files that are no longer needed.

2. Query the DBA_LOGMNR_PURGED_LOG view to list the archived redo log files that can be removed:

   ```
   SQL> SELECT * FROM DBA_LOGMNR_PURGED_LOG;

   FILE_NAME
   -----------------------------------
   /boston/arc_dest/arc_1_40_509538672.log
   /boston/arc_dest/arc_1_41_509538672.log
   /boston/arc_dest/arc_1_42_509538672.log
   /boston/arc_dest/arc_1_43_509538672.log
   /boston/arc_dest/arc_1_44_509538672.log
   /boston/arc_dest/arc_1_45_509538672.log
   /boston/arc_dest/arc_1_46_509538672.log
   /boston/arc_dest/arc_1_47_509538672.log
   ```

3. Use an operating system-specific command to delete the archived redo log files listed by the query.

10.5 Customizing a Logical Standby Database

This section contains the following topics:

- Customizing Logging of Events in the DBA_LOGSTDBY_EVENTS View
- Using DBMS_LOGSTDBY.SKIP to Prevent Changes to Specific Schema Objects
- Setting up a Skip Handler for a DDL Statement
- Modifying a Logical Standby Database
- Adding or Re-Creating Tables On a Logical Standby Database

> **See Also:** The DBMS_LOGSTDBY package in *Oracle Database PL/SQL Packages and Types Reference*

10.5.1 Customizing Logging of Events in the DBA_LOGSTDBY_EVENTS View

The DBA_LOGSTDBY_EVENTS view can be thought of as a circular log containing the most recent interesting events that occurred in the context of SQL Apply. By default the last 10,000 events are remembered in the event view. You can change the number of events logged by invoking the DBMS_LOGSTDBY.APPLY_SET procedure. For example, to ensure that the last 100,000 events are recorded, you can issue the following statement:

```
SQL> EXECUTE DBMS_LOGSTDBY.APPLY_SET ('MAX_EVENTS_RECORDED', '100000');
```

Errors that cause SQL Apply to stop are always recorded in the DBA_LOGSTDBY_EVENTS view (unless there is insufficient space in the SYSTEM tablespace). These events are always put into the alert 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.

The following examples show DBMS_LOGSTDBY subprograms that specify events to be recorded in the view.

Example 1 Determining If DDL Statements Have Been Applied

For example, to record applied DDL transactions to the DBA_LOGSTDBY_EVENTS view, issue the following statement:

```
SQL> EXECUTE DBMS_LOGSTDBY.APPLY_SET ('RECORD_APPLIED_DDL', 'TRUE');
```

Example 2 Checking the DBA_LOGSTDBY_EVENTS View for Unsupported Operations

To capture information about transactions running on the primary database that will not be supported by a logical standby database, issue the following statement:

```
SQL> EXEC DBMS_LOGSTDBY.APPLY_SET('RECORD_UNSUPPORTED_OPERATIONS', 'TRUE');
```

Then, check the DBA_LOGSTDBY_EVENTS view for any unsupported operations. Usually, an operation on an unsupported table is silently ignored by SQL Apply. However, during rolling upgrade (while the standby database is at a higher version and mining redo generated by a lower versioned primary database), if you performed an unsupported operation on the primary database, the logical standby database may not be the one to which you want to perform a switchover. Data Guard will log at least one unsupported operation per table in the DBA_LOGSTDBY_EVENTS view. Chapter 12, "Using SQL Apply to Upgrade the Oracle Database" provides detailed information about rolling upgrades.

10.5.2 Using DBMS_LOGSTDBY.SKIP to Prevent Changes to Specific Schema Objects

By default, all supported tables in the primary database are replicated in the logical standby database. You can change the default behavior by specifying rules to skip applying modifications to specific tables. For example, to omit changes to the HR.EMPLOYEES table, you can specify rules to prevent application of DML and DDL changes to the specific table. For example:

1. Stop SQL Apply:

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

2. Register the SKIP rules:

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP (stmt => 'DML', schema_name => 'HR', -
           object_name => 'EMPLOYEES');
SQL> EXECUTE DBMS_LOGSTDBY.SKIP (stmt => 'SCHEMA_DDL', schema_name => 'HR', -
           object_name => 'EMPLOYEES');
```

3. Start SQL Apply:

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

10.5.3 Setting up a Skip Handler for a DDL Statement

You can create a procedure to intercept certain DDL statements and replace the original DDL statement with a different one. For example, if the file system organization in the logical standby database is different than that in the primary database, you can write a DBMS_LOGSTDBY.SKIP procedure to transparently handle DDL transactions with file specifications.

The following procedure can handle different file system organization between the primary database and standby database, as long as you use a specific naming convention for your file-specification string.

1. Create the skip procedure to handle tablespace DDL transactions:

```
CREATE OR REPLACE PROCEDURE SYS.HANDLE_TBS_DDL (
  OLD_STMT  IN  VARCHAR2,
  STMT_TYP  IN  VARCHAR2,
  SCHEMA    IN  VARCHAR2,
  NAME      IN  VARCHAR2,
  XIDUSN    IN  NUMBER,
  XIDSLT    IN  NUMBER,
  XIDSQN    IN  NUMBER,
  ACTION    OUT NUMBER,
  NEW_STMT  OUT VARCHAR2
) AS
BEGIN

-- All primary file specification that contains a directory
-- /usr/orcl/primary/dbs
-- should go to /usr/orcl/stdby directory specification


  NEW_STMT := REPLACE(OLD_STMT,
                      '/usr/orcl/primary/dbs',
                      '/usr/orcl/stdby');

  ACTION := DBMS_LOGSTDBY.SKIP_ACTION_REPLACE;

EXCEPTION
```

```
  WHEN OTHERS THEN
    ACTION := DBMS_LOGSTDBY.SKIP_ACTION_ERROR;
    NEW_STMT := NULL;
END HANDLE_TBS_DDL;
```

2. Stop SQL Apply:

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

3. Register the skip procedure with SQL Apply:

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP (stmt => 'TABLESPACE', -
             proc_name => 'sys.handle_tbs_ddl');
```

4. Start SQL Apply:

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

10.5.4 Modifying a Logical Standby Database

Logical standby databases can be used for reporting activities, even while SQL statements are being applied. The *database guard* controls user access to tables in a logical standby database, and the `ALTER SESSION DISABLE GUARD` statement is used to bypass the database guard and allow modifications to the tables in the logical standby database.

Note: To use a logical standby database to host other applications that process data being replicated from the primary database while creating other tables of their own, the database guard must be set to `STANDBY`. For such applications to work seamlessly, make sure that you are running with `PRESERVE_COMMIT_ORDER` set to `TRUE` (the default setting for SQL Apply). (See *Oracle Database PL/SQL Packages and Types Reference* for information about the `PRESERVE_COMMIT_ORDER` parameter in the `DBMS_LOGSTDBY` PL/SQL package.)

Issue the following SQL statement to set the database guard to `STANDBY`:

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

Under this guard setting, tables being replicated from the primary database are protected from user modifications, but tables created on the standby database can be modified by the applications running on the logical standby.

By default, a logical standby database operates with the database guard set to `ALL`, which is its most restrictive setting, and does not allow any user changes to be performed to the database. You can override the database guard to allow changes to the logical standby database by executing the `ALTER SESSION DISABLE GUARD` statement. Privileged users can issue this statement to turn the database guard off for the current session.

The following sections provide some examples. The discussions in these sections assume that the database guard is set to `ALL` or `STANDBY`.

10.5.4.1 Performing DDL on a Logical Standby Database

This section describes how to add a constraint to a table maintained through SQL Apply.

By default, only accounts with SYS privileges can modify the database while the database guard is set to ALL or STANDBY. If you are logged in as SYSTEM or another privileged account, you will not be able to issue DDL statements on the logical standby database without first bypassing the database guard for the session.

The following example shows how to stop SQL Apply, bypass the database guard, execute SQL statements on the logical standby database, and then reenable the guard. In this example, a soundex index is added to the surname column of SCOTT.EMP in order to speed up partial match queries. A soundex index could be prohibitive to maintain on the primary server.

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

SQL> ALTER SESSION DISABLE GUARD;
PL/SQL procedure successfully completed.

SQL> CREATE INDEX EMP_SOUNDEX ON SCOTT.EMP(SOUNDEX(ENAME));
Table altered.

SQL> ALTER SESSION ENABLE GUARD;
PL/SQL procedure successfully completed.

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

SQL> SELECT ENAME,MGR FROM SCOTT.EMP WHERE SOUNDEX(ENAME) = SOUNDEX('CLARKE');

ENAME            MGR
----------       ----------
CLARK             7839
```

Oracle recommends that you do not perform DML operations on tables maintained by SQL Apply while the database guard bypass is enabled. This will introduce deviations between the primary and standby databases that will make it impossible for the logical standby database to be maintained.

10.5.4.2 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 the application is to perform reporting activities, the application might need to issue DML (insert, update, and delete) operations on a logical standby database. It might even need to create or drop tables.

You can set up the database guard to allow reporting operations to modify data as long as the data is not being maintained through SQL Apply. To do this, you must:

- Specify the set of tables on the logical standby database to which an application can write data by executing the DBMS_LOGSTDBY.SKIP procedure. Skipped tables are not maintained through SQL Apply.
- Set the database guard to protect only standby tables.

In the following example, it is assumed that the tables to which the report is writing are also on the primary database.

The example stops SQL Apply, skips the tables, and then restarts SQL Apply. The reporting application will be able to write to TESTEMP% in HR. They will no longer be maintained through SQL Apply.

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

SQL> EXECUTE DBMS_LOGSTDBY.SKIP(stmt => 'SCHEMA_DDL',-
     schema_name => 'HR', -
     object_name => 'TESTEMP%');
PL/SQL procedure successfully completed.

SQL> EXECUTE DBMS_LOGSTDBY.SKIP('DML','HR','TESTEMP%');
PL/SQL procedure successfully completed.

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

Once SQL Apply starts, it needs to update metadata on the standby database for the newly specified tables added in the skip rules. Attempts to modify the newly skipped table until SQL Apply has had a chance to update the metadata will fail. You can find out if SQL Apply has successfully taken into account the `SKIP` rule you just added by issuing the following query:

```
SQL> SELECT VALUE FROM DBA_LOGSDTBY_PARAMETERS
     WHERE NAME = 'GUARD_STANDBY';

VALUE
---------------
Ready
```

Once the `VALUE` column displays "`Ready`" SQL Apply has successfully updated all relevant metadata for the skipped table, and it is safe to modify the table.

> **See Also:** Section C.11, "DDL Statements Supported by a Logical Standby Database" and the `DBMS_LOGSTDBY` package in *Oracle Database PL/SQL Packages and Types Reference*

10.5.5 Adding or Re-Creating Tables On a Logical Standby Database

Typically, you use the `DBMS_LOGSTDBY.INSTANTIATE_TABLE` procedure to re-create a table after an unrecoverable operation. You can also use this procedure to enable SQL Apply on a table that was formerly skipped.

Before you can create a table, it must meet the requirements described in Section 4.1.2, "Ensure Table Rows in the Primary Database Can Be Uniquely Identified". Then, you can use the following steps to re-create a table named `HR.EMPLOYEES` and resume SQL Apply. The directions assume that there is already a database link `BOSTON` defined to access the primary database.

The following list shows how to re-create a table and restart SQL Apply on that table:

1. Stop SQL Apply:

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

2. Ensure no operations are being skipped for the table in question by querying the `DBA_LOGSTDBY_SKIP` view:

   ```
   SQL> SELECT * FROM DBA_LOGSTDBY_SKIP;
   ERROR  STATEMENT_OPT        OWNER          NAME                PROC
   -----  -------------------  -------------  ----------------    -----
   N      SCHEMA_DDL           HR             EMPLOYEES
   N      DML                  HR             EMPLOYEES
   N      SCHEMA_DDL           OE             TEST_ORDER
   ```

```
N      DML                  OE             TEST_ORDER
```

Because you already have skip rules associated with the table that you want to re-create on the logical standby database, you must first delete those rules. You can accomplish that by calling the DBMS_LOGSTDBY.UNSKIP procedure. For example:

```
SQL> EXECUTE DBMS_LOGSTDBY.UNSKIP(stmt => 'DML', -
     schema_name => 'HR', -
     object_name => 'EMPLOYEES');

SQL> EXECUTE DBMS_LOGSTDBY.UNSKIP(stmt => 'SCHEMA_DDL', -
     schema_name => 'HR', -
     object_name => 'EMPLOYEES');
```

3. Re-create the table HR.EMPLOYEES with all its data in the logical standby database by using the DBMS_LOGSTDBY.INSTANTIATE_TABLE procedure. For example:

```
SQL> EXECUTE DBMS_LOGSTDBY.INSTANTIATE_TABLE(schema_name => 'HR', -
     object_name => 'EMPLOYEES', -
     dblink => 'BOSTON');
```

4. Start SQL Apply:

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

> **See Also:** *Oracle Database PL/SQL Packages and Types Reference* for information about the DBMS_LOGSTDBY.UNSKIP and the DBMS_LOGSTDBY.INSTANTIATE_TABLE procedures

To ensure a consistent view across the newly instantiated table and the rest of the database, wait for SQL Apply to catch up with the primary database before querying this table. You can do this by performing the following steps:

1. On the primary database, determine the current SCN by querying the V$DATABASE view:

```
SQL> SELECT CURRENT_SCN FROM V$DATABASE@BOSTON;
CURRENT_SCN
--------------------
345162788
```

2. Make sure SQL Apply has applied all transactions committed before the CURRENT_SCN returned in the previous query:

```
SQL> SELECT APPLIED_SCN FROM V$LOGSTDBY_PROGRESS;

APPLIED_SCN
-------------------------
345161345
```

When the APPLIED_SCN returned in this query is greater than the CURRENT_SCN returned in the first query, it is safe to query the newly re-created table.

10.6 Managing Specific Workloads In the Context of a Logical Standby Database

This section contains the following topics:

- Importing a Transportable Tablespace to the Primary Database

- Using Materialized Views
- How Triggers and Constraints Are Handled on a Logical Standby Database
- Recovering Through the Point-in-Time Recovery Performed at the Primary

10.6.1 Importing a Transportable Tablespace to the Primary Database

Perform the following steps to import a tablespace to the primary database.

1. Disable the guard setting so that you can modify the logical standby database:

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

2. Import the tablespace at the logical standby database.
3. Enable the database guard setting:

```
SQL> ALTER DATABASE GUARD ALL;
```

4. Import the tablespace at the primary database.

10.6.2 Using Materialized Views

Logical Standby automatically skips DDL statements related to materialized views:

- `CREATE`, `ALTER`, or `DROP MATERIALIZED VIEW`
- `CREATE`, `ALTER` or `DROP MATERIALIZED VIEW LOG`

New materialized views that are created, altered, or dropped on the primary database after the logical standby database has been created will not be created on the logical standby database. However, materialized views created on the primary database prior to the logical standby database being created will be present on the logical standby database.

Logical Standby supports the creation and maintenance of new materialized views locally on the logical standby database in addition to other kinds of auxiliary data structure (see section 9.4.4). For example, online transaction processing (OLTP) systems frequently use highly normalized tables for update performance but these can lead to slower response times for complex decision support queries. Materialized views that denormalize the replicated data for more efficient query support on the logical standby database can be created, as follows:

```
SQL> ALTER SESSION DISABLE GUARD;

SQL> ALTER TABLE DEPT ADD (CONSTRAINT DEPT_PK PRIMARY KEY (DEPTNO));

SQL> ALTER TABLE  EMP ADD (CONSTRAINT EMP_FK FOREIGN KEY (DEPTNO)
  2  REFERENCES DEPT(DEPTNO));

SQL> CREATE MATERIALIZED VIEW LOG ON EMP
  2  WITH ROWID (EMPNO, ENAME, MGR, DEPTNO) INCLUDING NEW VALUES;

SQL> CREATE MATERIALIZED VIEW LOG ON DEPT
  2  WITH ROWID (DEPTNO, DNAME) INCLUDING NEW VALUES;

SQL> CREATE MATERIALIZED VIEW MANAGED_BY
  2  REFRESH ON DEMAND
  3  ENABLE QUERY REWRITE
  4  AS SELECT  E.ENAME, M.ENAME AS MANAGER
  5  FROM EMP E, EMP M WHERE E.MGR=M.EMPNO;
```

```
SQL> CREATE MATERIALIZED VIEW IN_DEPT
  2  REFRESH FAST ON COMMIT
  3  ENABLE QUERY REWRITE
  4  AS SELECT E.ROWID AS ERID, D.ROWID AS DRID, E.ENAME, D.DNAME
  5  FROM EMP E, DEPT D WHERE E.DEPTNO=D.DEPTNO;
```

On a logical standby database:

- An ON-COMMIT materialized view is refreshed automatically on the logical standby database when the transaction commit occurs.
- An ON-DEMAND materialized view is not automatically refreshed: the `DBMS_MVIEW.REFRESH` procedure must be executed to refresh it.

For example, issuing the following command would refresh the ON-DEMAND materialized view created in the previous example:

```
SQL> ALTER SESSION DISABLE GUARD;

SQL> EXECUTE DBMS_MVIEW.REFRESH (LIST => 'SCOTT.MANAGED_BY', METHOD => 'C');
```

If `DBMS_SCHEDULER` jobs are being used to periodically refresh on-demand materialized views, the database guard must be set to `STANDBY`. (It is not possible to use the `ALTER SESSION DISABLE GUARD` statement inside a PL/SQL block and have it take effect.)

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

By default, triggers and constraints are automatically enabled and handled on logical standby databases.

For triggers and constraints on tables *maintained* by SQL Apply:

- Constraints — Check constraints are evaluated on the primary database and do not need to be re-evaluated on the logical standby database.
- Triggers — The effects of the triggers executed on the primary database are logged and applied on the standby database. The exception to this rule is triggers whose `Fire_Once_Only` firing property is set to `FALSE`.

For triggers and constraints on tables *not maintained* by SQL Apply:

- Constraints are evaluated
- Triggers are fired

10.6.4 Using Triggers to Replicate Unsupported Tables

Tables that are unsupported due to simple object type columns can be replicated using non-Fire_Once_Only triggers. A regular DML trigger can be used on the primary to flatten the object type into a table that can be supported. A non-Fire_Once_Only trigger can be used on the logical standby to reconstitute the object type and update the unsupported table in a transactional manner.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for descriptions of the `DBMS_DDL.SET_TRIGGER_FIRING_PROPERTY` procedure and the `DBMS_LOGSTDBY.IS_APPLY_SERVER` function

The following example shows how a table with a simple object type could be replicated using triggers. This example shows how to handle inserts; the same principle can be applied to updating and deleting. Nested tables and VARRAYs can also be replicated using this technique with the additional step of a loop to normalize the nested data.

```
-- simple object type
create or replace type Person as object
(
  FirstName    varchar2(50),
  LastName     varchar2(50),
  BirthDate    Date
)

-- unsupported object table
create table employees
(
  IdNumber     varchar2(10) ,
  Department   varchar2(50),
  Info         Person
)

-- supported table populated via trigger
create table employees_transfer
(
  t_IdNumber   varchar2(10),
  t_Department varchar2(50),
  t_FirstName  varchar2(50),
  t_LastName   varchar2(50),
  t_BirthDate  Date
)
--
-- create this trigger to flatten object table on the primary
-- this trigger will not fire on the standby
--
create or replace trigger flatten_employees
  after insert on employees for each row
declare
begin
  insert into employees_transfer
    (t_IdNumber, t_Department, t_FirstName, t_LastName, t_BirthDate)
  values
    (:new.IdNumber, :new.Department,
 :new.Info.FirstName,:new.Info.LastName, :new.Info.BirthDate);
end

--
-- create this trigger at the logical standby database
-- to populate object table on the standby
-- this trigger only fires when apply replicates rows to the standby
--
create or replace trigger reconstruct_employees
  after insert on employees_transfer for each row
begin
  if dbms_logstdby.is_apply_server() then
    insert into employees (IdNumber, Department, Info)
    values (:new.t_IdNumber, :new.t_Department,
          Person(:new.t_FirstName, :new.t_LastName, :new.t_BirthDate));
  end if;
end
```

```
-- set this trigger to fire from the apply server
execute dbms_ddl.set_trigger_firing_property(trig_owner=> 'scott', trig_name=>
'reconstruct_employees', fire_once => FALSE);
```

10.6.5 Recovering Through the Point-in-Time Recovery Performed at the Primary

When a logical standby database receives a new branch of redo data, SQL Apply automatically takes the new branch of redo data. For logical standby databases, no manual intervention is required if the standby database did not apply redo data past the new resetlogs SCN (past the start of the new branch of redo data)

The following table describes how to resynchronize the standby database with the primary database branch.

If the standby database. . .	Then. . .	Perform these steps. . .
Has not applied redo data past the new resetlogs SCN (past the start of the new branch of redo data)	SQL Apply automatically takes the new branch of redo data.	No manual intervention is necessary. SQL Apply automatically resynchronizes the standby database with the new branch of redo data.
Has applied redo data past the new resetlogs SCN (past the start of the new branch of redo data) and Flashback Database is enabled on the standby database	The standby database is recovered *in the future* of the new branch of redo data.	1. Follow the procedure in Section 13.3.2, "Flashing Back a Logical Standby Database to a Specific Point-in-Time" to flash back a logical standby database. 2. Restart SQL Apply to continue application of redo onto the new reset logs branch. SQL Apply 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, "Creating a Logical Standby Database".
Is missing archived redo log files from the end of the previous branch of redo data	SQL Apply cannot continue until the missing log files are retrieved.	Locate and register missing archived redo log files from the previous branch.

See *Oracle Database Backup and Recovery User's Guide* for more information about database incarnations, recovering through an `OPEN RESETLOGS` operation, and Flashback Database.

10.7 Tuning a Logical Standby Database

This section contains the following topics:

- Create a Primary Key RELY Constraint
- Gather Statistics for the Cost-Based Optimizer
- Adjust the Number of Processes
- Adjust the Memory Used for LCR Cache
- Adjust How Transactions are Applied On the Logical Standby Database

10.7.1 Create a Primary Key RELY Constraint

On the primary database, if a table does not have a primary key or a unique index and you are certain the rows are unique, then create a primary key RELY constraint. On the logical standby database, create an index on the columns that make up the primary key. The following query generates a list of tables with no index information that can be used by a logical standby database to apply to uniquely identify rows. By creating an index on the following tables, performance can be improved significantly.

```
SQL> SELECT OWNER, TABLE_NAME FROM DBA_TABLES
  2> WHERE OWNER NOT IN (SELECT OWNER FROM DBA_LOGSTDBY_SKIP
  3> WHERE STATEMENT_OPT = 'INTERNAL SCHEMA')
  4> MINUS
  5> SELECT DISTINCT TABLE_OWNER, TABLE_NAME FROM DBA_INDEXES
  6> WHERE INDEX_TYPE NOT LIKE ('FUNCTION-BASED%')
  7> MINUS
  8> SELECT OWNER, TABLE_NAME FROM DBA_LOGSTDBY_UNSUPPORTED;
```

You can add a rely primary key constraint to a table on the primary database, as follows:

1. Add the primary key rely constraint at the primary database:

   ```
   SQL> ALTER TABLE HR.TEST_EMPLOYEES ADD PRIMARY KEY (EMPNO) RELY DISABLE;
   ```

   This will ensure that the EMPNO column, which can be used to uniquely identify the rows in HR.TEST_EMPLOYEES table, will be supplementally logged as part of any updates done on that table.

   Note that the HR.TEST_EMPLOYEES table still does not have any unique index specified on the logical standby database. This may cause UPDATE statements to do full table scans on the logical standby database. You can remedy that by adding a unique index on the EMPNO column on the logical standby database.

   See Section 4.1.2, "Ensure Table Rows in the Primary Database Can Be Uniquely Identified" and *Oracle Database SQL Language Reference* for more information about RELY constraints.

   Perform the remaining steps on the logical standby database.

2. Stop SQL Apply:

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

3. Disable the guard so that you can modify a maintained table on the logical standby database:

   ```
   SQL> ALTER SESSION DISABLE GUARD;
   ```

4. Add a unique index on EMPNO column:

   ```
   SQL> CREATE UNIQUE INDEX UI_TEST_EMP ON HR.TEST_EMPLOYEES (EMPNO);
   ```

5. Enable the guard:

   ```
   SQL> ALTER SESSION ENABLE GUARD;
   ```

6. Start SQL Apply:

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

10.7.2 Gather Statistics for the Cost-Based Optimizer

Statistics should be gathered on the standby database because the cost-based optimizer (CBO) uses them to determine the optimal query execution path. New statistics should be gathered after the data or structure of a schema object is modified in ways that make the previous statistics inaccurate. For example, after inserting or deleting a significant number of rows into a table, collect new statistics on the number of rows.

Statistics should be gathered on the standby database because DML and DDL operations on the primary database are executed as a function of the workload. While the standby database is logically equivalent to the primary database, SQL Apply might execute the workload in a different way. This is why using the STATS pack on the logical standby database and the `V$SYSSTAT` view can be useful in determining which tables are consuming the most resources and table scans.

See Also:

- Section 4.1.2, "Ensure Table Rows in the Primary Database Can Be Uniquely Identified"
- *Oracle Database SQL Language Reference* for more information about `RELY` constraints

10.7.3 Adjust the Number of Processes

The following sections describe:

- Adjusting the Number of APPLIER Processes
- Adjusting the Number of PREPARER Processes

There are three parameters that can be modified to control the number of processes allocated to SQL Apply: MAX_SERVERS, APPLY_SERVERS, and PREPARE_SERVERS. The following relationships must always hold true:

- `APPLY_SERVERS + PREPARE_SERVERS = MAX_SERVERS - 3`

  This is because SQL Apply always allocates one process for the `READER`, `BUILDER`, and `ANALYZER` roles.

- By default, `MAX_SERVERS` is set to 9, `PREPARE_SERVERS` is set to 1, and `APPLY_SERVERS` is set to 5.

- Oracle recommends that you only change the `MAX_SERVERS` parameter through the `DBMS_LOGSTDBY.APPLY_SET` procedure, and allow SQL Apply to distribute the server processes appropriately between prepare and apply processes.

- SQL Apply uses a process allocation algorithm that allocates 1 `PREPARE_SERVER` for every 20 server processes allocated to SQL Apply as specified by `MAX_SERVER` and limits the number of `PREPARE_SERVERS` to 5. Thus, if you set `MAX_SERVERS` to any value between 1 and 20, SQL Apply allocates 1 server process to act as a `PREPARER`, and allocates the rest of the processes as `APPLIERS` while satisfying the relationship previously described. Similarly, if you set `MAX_SERVERS` to a value between 21 and 40, SQL Apply allocates 2 server processes to act as `PREPARERS` and the rest as `APPLIERS`, while satisfying the relationship previously described. You can override this internal process allocation algorithm by setting `APPLY_SERVERS` and `PREPARE_SERVERS` directly, provided that the previously described relationship is satisfied.

10.7.3.1 Adjusting the Number of APPLIER Processes

Perform the following steps to find out whether adjusting the number of APPLIER processes will help you achieve greater throughput:

1. Determine if APPLIER processes are busy by issuing the following query:

```
SQL> SELECT COUNT(*) AS IDLE_APPLIER
     FROM V$LOGSTDBY_PROCESS
     WHERE TYPE = 'APPLIER' and status_code = 16166;

IDLE_APPLIER
------------------------
0
```

2. Once you are sure there are no idle APPLIER processes, issue the following query to ensure there is enough work available for additional APPLIER processes if you choose to adjust the number of APPLIERS:

```
SQL> SELECT NAME, VALUE FROM V$LOGSTDBY_STATS WHERE NAME LIKE
'transactions%';
```

   These two statistics keep a cumulative total of transactions that are ready to be applied by the APPLIER processes and the number of transactions that have already been applied.

   If the number (`transactions mined - transactions applied`) is higher than twice the number of APPLIER processes available, an improvement in throughput is possible if you increase the number of APPLIER processes.

   Note: The number is a rough measure of ready work. The workload may be such that an interdependency between ready transactions will prevent additional available APPLIER processes from applying them. For instance, if the majority of the transactions that are ready to be applied are DDL transactions, adding more APPLIER processes will not result in a higher throughput.

   Suppose you want to adjust the number of APPLIER processes to 20 from the default value of 5, while keeping the number of PREPARER processes to 1. Because you have to satisfy the following equation:

```
APPLY_SERVERS + PREPARE_SERVERS = MAX_SERVERS - 3
```

   you will first need to set MAX_SERVERS to 24. Once you have done that, you can then set the number of APPLY_SERVERS to 20, as follows:

```
SQL> EXECUTE DBMS_LOGSTDBY.APPLY_SET('MAX_SERVERS', 24);
SQL> EXECUTE DBMS_LOGSTDBY.APPLY_SET('APPLY_SERVERS', 20);
```

10.7.3.2 Adjusting the Number of PREPARER Processes

In only rare cases do you need to adjust the number of PREPARER processes. Before you decide to increase the number of PREPARER processes, ensure the following conditions are true:

- All PREPARER processes are busy
- The number of transactions ready to be applied is less than the number of APPLIER processes available
- There are idle APPLIER processes

The following steps show how to determine these conditions are true:

1. Ensure all PREPARER processes are busy:

```
SQL> SELECT COUNT(*) AS IDLE_PREPARER
     FROM V$LOGSTDBY_PROCESS
     WHERE TYPE = 'PREPARER' and status_code = 16166;
IDLE_PREPARER
-------------
0
```

2. Ensure the number of transactions ready to be applied is less than the number of APPLIER processes:

```
SQL> SELECT NAME, VALUE FROM V$LOGSTDBY_STATS
     WHERE NAME LIKE 'transactions%';
NAME                          VALUE
---------------------         -------
transactions ready             27896
transactions applied           27892

SQL> SELECT COUNT(*) AS APPLIER_COUNT
     FROM V$LOGSTDBY_PROCESS WHERE TYPE = 'APPLIER';
APPLIER_COUNT
-------------
20
```

   Note: Issue this query several times to ensure this is not a transient event.

3. Ensure there are idle APPLIER processes:

```
SQL> SELECT COUNT(*) AS IDLE_APPLIER
     FROM V$LOGSTDBY_PROCESS
     WHERE TYPE = 'APPLIER' and status_code = 16166;
IDLE_APPLIER
------------------------
19
```

In the example, all three conditions necessary for increasing the number of PREPARER processes have been satisfied. Suppose you want to keep the number of APPLIER processes set to 20, and increase the number of PREPARER processes from 1 to 3. Because you always have to satisfy the following equation:

```
APPLY_SERVERS + PREPARE_SERVERS = MAX_SERVERS - 3
```

you will first need to increase the number MAX_SERVERS from 24 to 26 to accommodate the increased number of preparers. You can then increase the number of PREPARER processes, as follows:

```
SQL> EXECUTE DBMS_LOGSTDBY.APPLY_SET('MAX_SERVERS', 26);
SQL> EXECUTE DBMS_LOGSTDBY.APPLY_SET('PREPARE_SERVERS', 3);
```

10.7.4 Adjust the Memory Used for LCR Cache

For some workloads, SQL Apply may use a large number of pageout operations, thereby reducing the overall throughput of the system. To find out whether increasing memory allocated to LCR cache will be beneficial, perform the following steps:

1. Issue the following query to obtain a snapshot of pageout activity:

```
SQL> SELECT NAME, VALUE FROM V$LOGSTDBY_STATS WHERE NAME LIKE '%page%'
OR NAME LIKE '%uptime%' OR NAME LIKE '%idle%';
```

```
NAME                          VALUE
---------------------------   ---------------
coordinator uptime in secs              894856
bytes paged out                          20000
seconds spent in pageout                     2
system idle time in secs                  1000
```

2. Issue the query again in 5 minutes:

```
SQL> SELECT NAME, VALUE FROM V$LOGSTDBY_STATS WHERE NAME LIKE '%page%'
OR NAME LIKE '%uptime%' OR NAME LIKE '%idle%';

NAME                          VALUE
---------------------------   ---------------
coordinator uptime in secs              895156
bytes paged out                        1020000
seconds spent in pageout                   100
system idle time in secs                  1000
```

3. Compute the normalized pageout activity. For example:

```
Change in coordinator uptime (C)= (895156 - 894856) = 300 secs
Amount of additional idle time (I)= (1000 - 1000) = 0
Change in time spent in pageout (P) = (100 - 2) = 98 secs
Pageout time in comparison to uptime = P/(C-I) = 98/300 ~ 32.67%
```

Ideally, the pageout activity should not consume more than 5 percent of the total uptime. If you continue to take snapshots over an extended interval and you find the pageout activities continue to consume a significant portion of the apply time, increasing the memory size may provide some benefits. You can increase the memory allocated to SQL Apply by setting the memory allocated to LCR cache (for this example, the SGA is set to 1 GB):

```
SQL> EXECUTE DBMS_LOGSTDBY.APPLY_SET('MAX_SGA', 1024);
PL/SQL procedure successfully completed
```

10.7.5 Adjust How Transactions are Applied On the Logical Standby Database

By default transactions are applied on the logical standby database in the exact order in which they were committed on the primary database. The strict default order of committing transactions allow any application to run transparently on the logical standby database.

However, many applications do not require such strict ordering among all transactions. Such applications do not require transactions containing non-overlapping sets of rows to be committed in the same order that they were committed at the primary database. This less strict ordering typically results in higher apply rates at the logical standby database. You can change the default order of committing transactions by performing the following steps:

1. Stop SQL Apply:

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

2. Issue the following to allow transactions to be applied out of order from how they were committed on the primary databases:

```
SQL> EXECUTE DBMS_LOGSTDBY.APPLY_SET('PRESERVE_COMMIT_ORDER', 'FALSE');
PL/SQL procedure successfully completed
```

3. Start SQL Apply:

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

Once you have caught up with the primary database (verify this by querying the V$LOGSTDBY_STATS view), and you are ready to open the logical standby database for reporting applications, you can change the apply mode as follows:

1. Stop SQL Apply:

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

2. Restore the default value for the PRESERVE_COMMIT_ORDER parameter:

```
SQL> EXECUTE DBMS_LOGSTDBY.APPLY_UNSET('PRESERVE_COMMIT_ORDER');
PL/SQL procedure successfully completed
```

3. Start SQL Apply:

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

For a typical online transaction processing (OLTP) workload, the nondefault mode can provide a 50 percent or better throughput improvement over the default apply mode.

10.8 Backup and Recovery in the Context of a Logical Standby Database

You can back up your logical standby database using the traditional methods available and then recover it by restoring the database backup and performing media recovery on the archived logs, in conjunction with the backup. The following items are relevant in the context of a logical standby database.

Considerations When Creating and Using a Local RMAN Recovery Catalog

If you plan to create the RMAN recovery catalog or perform any RMAN activity that modifies the catalog, you must be running with GUARD set to STANDBY at the logical standby database.

You can leave GUARD set to ALL, if the local recovery catalog is kept only in the logical standby control file.

Considerations For Control File Backup

Oracle recommends that you take a control file backup immediately after instantiating a logical standby database.

Considerations For Point-in-Time Recovery

When SQL Apply is started for the first time following point-in-time recovery, it must be able to either find the required archived logs on the local system or to fetch them from the primary database. Use the V$LOGSTDBY_PROCESS view to determine if any archived logs need to be restored on the primary database.

Considerations For Tablespace Point-in-Time Recovery

If you perform point-in-time recovery for a tablespace in a logical standby database, you must ensure one of the following:

- The tablespace contains no tables or partitions that are being maintained by the SQL Apply process

- If the tablespace contains tables or partitions that are being maintained by the SQL Apply process, you should either use the `DBMS_LOGSTDBY.INSTANTIATE_TABLE` procedure to reinstantiate all of the maintained tables contained in the recovered tablespace at the logical standby database, or use `DBMS_LOGSTDBY.SKIP` procedure to register all tables contained in the recovered tablespace to be skipped from the maintained table list at the logical standby database.

11

Using RMAN to Back Up and Restore Files

This chapter describes backup strategies using Oracle Recovery Manager (RMAN) with Data Guard and standby databases. RMAN can perform backups with minimal effect on the primary database and quickly recover from the loss of individual datafiles, or the entire database. RMAN and Data Guard can be used together to simplify the administration of a Data Guard configuration.

This chapter contains the following topics:

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.

See Also:

- *Oracle Database Backup and Recovery User's Guide* for more information about RMAN concepts and about using RMAN in a Data Guard environment
- *Oracle Database Backup and Recovery Reference* for detailed information about all RMAN commands used in this chapter

11.1 About RMAN File Management in a Data Guard Configuration

RMAN uses a recovery catalog to track filenames for all database files in a Data Guard environment. A recovery catalog is a database schema used by RMAN to store metadata about one or more Oracle databases. The catalog also records where the

online redo logs, standby redo logs, tempfiles, archived redo logs, backup sets, and image copies are created.

11.1.1 Interchangeability of Backups in a Data Guard Environment

RMAN commands use the recovery catalog metadata to behave transparently across different physical databases in the Data Guard environment. For example, you can back up a tablespace on a physical standby database and restore and recover it on the primary database. Similarly, you can back up a tablespace on a primary database and restore and recover it on a physical standby database.

> **Note:** Backups of logical standby databases are not usable at the primary database.

Backups of standby control files and nonstandby control files are interchangeable. For example, you can restore a standby control file on a primary database and a primary control file on a physical standby database. This interchangeability means that you can offload control file backups to one database in a Data Guard environment. RMAN automatically updates the filenames for database files during restore and recovery at the databases.

11.1.2 Association of Backups in a Data Guard Environment

The recovery catalog tracks the files in the Data Guard environment by associating every database file or backup file with a `DB_UNIQUE_NAME`. The database that creates a file is associated with the file. For example, if RMAN backs up the database with the unique name of `standby1`, then `standby1` is associated with this backup. A backup remains associated with the database that created it unless you use the `CHANGE ... RESET DB_UNIQUE_NAME` to associate the backup with a different database.

11.1.3 Accessibility of Backups in a Data Guard Environment

The accessibility of a backup is different from its association. In a Data Guard environment, the recovery catalog considers disk backups as accessible only to the database with which it is associated, whereas tape backups created on one database are accessible to all databases. If a backup file is not associated with any database, then the row describing it in the recovery catalog view shows `null` for the `SITE_KEY` column. By default, RMAN associates files whose `SITE_KEY` is `null` with the target database.

RMAN commands such as `BACKUP`, `RESTORE`, and `CROSSCHECK` work on any accessible backup. For example, for a `RECOVER COPY` operation, RMAN considers only image copies that are associated with the database as eligible to be recovered. RMAN considers the incremental backups on disk and tape as eligible to recover the image copies. In a database recovery, RMAN considers only the disk backups associated with the database and all files on tape as eligible to be restored.

To illustrate the differences in backup accessibility, assume that databases `prod` and `standby1` reside on different hosts. RMAN backs up datafile 1 on `prod` to `/prmhost/disk1/df1.dbf` on the production host and also to tape. RMAN backs up datafile 1 on `standby1` to `/sbyhost/disk2/df1.dbf` on the standby host and also to tape. If RMAN is connected to database `prod`, then you cannot use RMAN commands to perform operations with the `/sbyhost/disk2/df1.dbf` backup located on the standby host. However, RMAN does consider the tape backup made on `standby1` as eligible to be restored.

Note: You can FTP a backup from a standby host to a primary host or vice versa, connect as TARGET to the database on this host, and then CATALOG the backup. After a file is cataloged by the target database, the file is associated with the target database.

11.2 About RMAN Configuration in a Data Guard Environment

In a Data Guard configuration, the process of backing up control files, datafiles, and archived logs can be offloaded to the standby system, thereby minimizing the effect of backups on the production system. These backups can be used to recover the primary or standby database.

RMAN uses the DB_UNIQUE_NAME initialization parameter to distinguish one database site from another database site. Thus, it is critical that the uniqueness of DB_UNIQUE_NAME be maintained in a Data Guard configuration.

Only the primary database must be explicitly registered using the RMAN REGISTER DATABASE command. You do this after connecting RMAN to the recovery catalog and primary database as target.

Use the RMAN CONFIGURE command to set the RMAN configurations. When the CONFIGURE command is used with the FOR DB_UNIQUE_NAME option, it sets the RMAN site-specific configuration for the database with the DB_UNIQUE_NAME you specify.

For example, after connecting to the recovery catalog, you could use the following commands at an RMAN prompt to set the default device type to SBT for the BOSTON database that has a DBID of 1625818158. The RMAN SET DBID command is required only if you are not connected to a database as target.

```
SET DBID 1625818158;
CONFIGURE DEFAULT DEVICE TYPE TO SBT FOR DB_UNIQUE_NAME BOSTON;
```

11.3 Recommended RMAN and Oracle Database Configurations

This section describes the following RMAN and Oracle Database configurations, each of which can simplify backup and recovery operations:

- Oracle Database Configurations on Primary and Standby Databases
- RMAN Configurations at the Primary Database
- RMAN Configurations at a Standby Database Where Backups are Performed
- RMAN Configurations at a Standby Where Backups Are Not Performed

Configuration Assumptions

The configurations described in this section make the following assumptions:

- The standby database is a physical standby database, and backups are taken only on the standby database. See Section 11.9.1 for procedural changes if backups are taken on both primary and standby databases.
- An RMAN recovery catalog is required so that backups taken on one database server can be restored to another database server. It is not sufficient to use only the control file as the RMAN repository because the primary database will have no knowledge of backups taken on the standby database.

The RMAN recovery catalog organizes backup histories and other recovery-related metadata in a centralized location. The recovery catalog is configured in a database and maintains backup metadata. A recovery catalog does not have the space limitations of the control file and can store more historical data about backups.

A catalog server, physically separate from the primary and standby sites, is recommended in a Data Guard configuration because a disaster at either site will not affect the ability to recover the latest backups.

> **See Also:** *Oracle Database Backup and Recovery User's Guide* for more information about managing a recovery catalog

- All databases in the configuration use Oracle Database 11*g* Release 1 (11.1).
- Oracle Secure Backup software or 3rd-party media management software is configured with RMAN to make backups to tape.

11.3.1 Oracle Database Configurations on Primary and Standby Databases

The following Oracle Database configurations are recommended on every primary and standby database in the Data Guard environment:

- Configure a flash recovery area for each database (the recovery area is local to a database).

  The flash recovery area is a single storage location on a file system or Automatic Storage Management (ASM) disk group where all files needed for recovery reside. These files include the control file, archived logs, online redo logs, flashback logs, and RMAN backups. As new backups and archived logs are created in the flash recovery area, older files (which are either outside of the retention period, or have been backed up to tertiary storage) are automatically deleted to make room for them. In addition, notifications can be set up to alert the DBA when space consumption in the flash recovery area is nearing its predefined limit. The DBA can then take action, such as increasing the recovery area space limit, adding disk hardware, or decreasing the retention period.

  Set the following initialization parameters to configure the flash recovery area:

  ```
  DB_RECOVERY_FILE_DEST = <mount point or ASM Disk Group>
  DB_RECOVERY_FILE_DEST_SIZE = <disk space quota>
  ```

  > **See Also:** *Oracle Database Backup and Recovery User's Guide* for more information about configuring a flash recovery area

- Use a server parameter file (SPFILE) so that it can be backed up to save instance parameters in backups.
- Enable Flashback Database on primary and standby databases.

  When Flashback Database is enabled, Oracle Database maintains flashback logs in the flash recovery area. These logs can be used to roll the database back to an earlier point in time, without requiring a complete restore.

  > **See Also:** *Oracle Database Backup and Recovery User's Guide* for more information about enabling Flashback Database

11.3.2 RMAN Configurations at the Primary Database

To simplify ongoing use of RMAN, you can set a number of persistent configuration settings for each database in the Data Guard environment. These settings control many aspects of RMAN behavior. For example, you can configure the backup retention policy, default destinations for backups to tape or disk, default backup device type, and so on. You can use the `CONFIGURE` command to set and change RMAN configurations. The following RMAN configurations are recommended at the primary database:

1. Connect RMAN to the primary database and recovery catalog.
2. Configure the retention policy for the database as *n* days:

   ```
   CONFIGURE RETENTION POLICY TO RECOVERY WINDOW OF <n> DAYS;
   ```

   This configuration lets you keep the backups necessary to perform database recovery to any point in time within the specified number of days.

   Use the `DELETE OBSOLETE` command to delete any backups that are not required (per the retention policy in place) to perform recovery within the specified number of days.
3. Specify when archived logs can be deleted with the `CONFIGURE ARCHIVELOG DELETION POLICY` command. For example, if you want to delete logs after ensuring that they *shipped* to all destinations, use the following configuration:

   ```
   CONFIGURE ARCHIVELOG DELETION POLICY TO SHIPPED TO ALL STANDBY;
   ```

   If you want to delete logs after ensuring that they were *applied* on all standby destinations, use the following configuration:

   ```
   CONFIGURE ARCHIVELOG DELETION POLICY TO APPLIED ON ALL STANDBY;
   ```

4. Configure the connect string for the primary database and all standby databases, so that RMAN can connect remotely and perform resynchronization when the `RESYNC CATALOG FROM DB_UNIQUE_NAME` command is used. When you connect to the target instance, you must provide a net service name. This requirement applies even if the other database instance from where the resynchronization is done is on the local host. The target and remote instances must use the same `SYSDBA` password, which means that both instances must already have password files. You can create the password file with a single password so you can start all the database instances with that password file. For example, if the TNS alias to connect to a standby in Boston is `boston_conn_str`, you can use the following command to configure the connect identifier for the `BOSTON` database site:

   ```
   CONFIGURE DB_UNIQUE_NAME BOSTON CONNECT IDENTIFIER 'boston_conn_str';
   ```

   Note that the `'boston_conn_str'` does not include a username and password. It contains only the Oracle Net service name that can be used from any database site to connect to the `BOSTON` database site.

   After connect identifiers are configured for all standby databases, you can verify the list of standbys by using the `LIST DB_UNIQUE_NAME OF DATABASE` command.

See Also:

- *Oracle Database Backup and Recovery User's Guide* for more information about RMAN configurations
- *Oracle Database Backup and Recovery Reference* for more information about the RMAN `CONFIGURE` command

11.3.3 RMAN Configurations at a Standby Database Where Backups are Performed

The following RMAN configurations are recommended at a standby database where backups are done:

1. Connect RMAN to the standby database (where backups are performed) as target, and to the recovery catalog.
2. Enable automatic backup of the control file and the server parameter file:

   ```
   CONFIGURE CONTROLFILE AUTOBACKUP ON;
   ```

3. Skip backing up datafiles for which there already exists a valid backup with the same checkpoint:

   ```
   CONFIGURE BACKUP OPTIMIZATION ON;
   ```

4. Configure the tape channels to create backups as required by media management software:

   ```
   CONFIGURE CHANNEL DEVICE TYPE SBT PARMS '<channel parameters>';
   ```

5. Specify when the archived logs can be deleted with the `CONFIGURE ARCHIVELOG DELETION POLICY` command.

   Because the logs are backed up at the standby site, it is recommended that you configure the `BACKED UP` option for the log deletion policy.

   See Also: *Oracle Database Backup and Recovery User's Guide* for more information about enabling deletion policies for archived redo logs

11.3.4 RMAN Configurations at a Standby Where Backups Are Not Performed

The following RMAN configurations are recommended at a standby database where backups are *not* done:

1. Connect RMAN to the standby database as target, and to the recovery catalog.
2. Enable automatic deletion of archived logs once they are applied at the standby database:

   ```
   CONFIGURE ARCHIVELOG DELETION POLICY TO APPLIED ON ALL STANDBY;
   ```

11.4 Backup Procedures

This section describes the RMAN scripts and procedures used to back up Oracle Database in a Data Guard configuration. The following topics are covered:

- Using Disk as Cache for Tape Backups
- Performing Backups Directly to Tape

Note: Oracle's Maximum Availability Architecture (MAA) best practices recommend that backups be taken at both the primary and the standby databases to reduce MTTR, in case of double outages and to avoid introducing new site practices upon switchover and failover.

Backups of Server Parameter Files

Prior to Oracle Database 11*g*, backups of server parameter files (SPFILEs) were assumed to be usable at any other standby database. However, in practice, it is not possible for all standby databases to use the same SPFILE. To address this problem, RMAN does not allow an SPFILE backup taken at one database site to be used at another database site. This restriction is in place only when the `COMPATIBLE` initialization parameter is set to 11.0.0.

The standby database allows you to offload all backup operations to one specific standby database, except the backups of SPFILE. However, if the `COMPATIBLE` initialization parameter is set to 11.0.0, the SPFILE can be backed up to disk and cataloged manually at standby sites where backups are written to tape. The additional metadata stored in SPFILE backup sets enables RMAN to identify which database SPFILE is contained in which backup set. Thus, the appropriate SPFILE backup is chosen during restore from tape.

11.4.1 Using Disk as Cache for Tape Backups

The flash recovery area on the standby database can serve as a disk cache for tape backup. Disk is used as the primary storage for backups, with tape providing long term, archival storage. Incremental tape backups are taken daily and full tape backups are taken weekly. The commands used to perform these backups are described in the following sections.

11.4.1.1 Commands for Daily Tape Backups Using Disk as Cache

When deciding on your backup strategy, Oracle recommends that you take advantage of daily incremental backups. Datafile image copies can be rolled forward with the latest incremental backups, thereby providing up-to-date datafile image copies at all times. RMAN uses the resulting image copy for media recovery just as it would use a full image copy taken at that system change number (SCN), without the overhead of performing a full image copy of the database every day. An additional advantage is that the time-to-recover is reduced because the image copy is updated with the latest block changes and fewer redo logs are required to bring the database back to the current state.

To implement daily incremental backups, a full database backup is taken on the first day, followed by an incremental backup on day two. Archived redo logs can be used to recover the database to any point in either day. For day three and onward, the previous day's incremental backup is merged with the datafile copy and a current incremental backup is taken, allowing fast recovery to any point within the last day. Redo logs can be used to recover the database to any point during the current day.

The script to perform daily backups looks as follows (the last line, `DELETE ARCHIVELOG ALL` is only needed if the flash recovery area is not used to store logs):

```
RESYNC CATALOG FROM DB_UNIQUE_NAME ALL;
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 SBT ARCHIVELOG ALL;
```

```
BACKUP BACKUPSET ALL;
DELETE ARCHIVELOG ALL;
```

The standby control file will be automatically backed up at the conclusion of the backup operation because the control file auto backup is enabled.

Explanations for what each command in the script does are as follows:

- `RESYNC CATALOG FROM DB_UNIQUE_NAME ALL`

  Resynchronizes the information from all other database sites (primary and other standby databases) in the Data Guard setup that are known to recovery catalog. For `RESYNC CATALOG FROM DB_UNIQUE_NAME` to work, RMAN should be connected to the target using the Oracle Net service name and all databases must use the same password file.

- `RECOVER COPY OF DATABASE WITH TAG 'OSS'`

  Rolls forward level 0 copy of the database by applying the level 1 incremental backup taken the day before. In the example script just shown, the previous day's incremental level 1 was tagged `OSS`. This incremental is generated by the `BACKUP DEVICE TYPE DISK ... DATABASE` command. On the first day this command is run there will be no roll forward because there is no incremental level 1 yet. A level 0 incremental will be created by the `BACKUP DEVICE TYPE DISK ... DATABASE` command. Again on the second day there is no roll forward because there is only a level 0 incremental. A level 1 incremental tagged `OSS` will be created by the `BACKUP DEVICE TYPE DISK ... DATABASE` command. On the third and following days, the roll forward will be performed using the level 1 incremental tagged `OSS` created on the previous day.

- `BACKUP DEVICE TYPE DISK INCREMENTAL LEVEL 1 FOR RECOVER OF COPY WITH TAG 'OSS' DATABASE`

  Create a new level 1 incremental backup. On the first day this command is run, this will be a level 0 incremental. On the second and following days, this will be a level 1 incremental.

- `BACKUP DEVICE TYPE SBT ARCHIVELOG ALL`

  Backs up archived logs to tape according to the deletion policy in place.

- `BACKUP BACKUPSET ALL`

  Backs up any backup sets created as a result of incremental backup creation.

- `DELETE ARCHIVELOG ALL`

  Deletes archived logs according to the log deletion policy set by the `CONFIGURE ARCHIVELOG DELETION POLICY` command. If the archived logs are in a flash recovery area, then they are automatically deleted when more open disk space is required. Therefore, you only need to use this command if you explicitly want to delete logs each day.

11.4.1.2 Commands for Weekly Tape Backups Using Disk as Cache

To back up all recovery-related files to tape, use the following command once a week:

```
BACKUP RECOVERY FILES;
```

This ensures that all current incremental, image copy, and archived log backups on disk are backed up to tape.

11.4.2 Performing Backups Directly to Tape

Oracle's Media Management Layer (MML) API lets third-party vendors build a media manager, software that works with RMAN and the vendor's hardware to allow backups to sequential media devices such as tape drives. A media manager handles loading, unloading, and labeling of sequential media such as tapes. You must install Oracle Secure Backup or third-party media management software to use RMAN with sequential media devices.

Take the following steps to perform backups directly to tape, by default:

1. Connect RMAN to the standby database (as the target database) and recovery catalog.
2. Execute the `CONFIGURE` command as follows:

```
CONFIGURE DEFAULT DEVICE TYPE TO SBT;
```

In this scenario, full backups are taken weekly, with incremental backups taken daily on the standby database.

> **See Also:** *Oracle Database Backup and Recovery User's Guide* for more information about how to configure RMAN for use with a media manager

11.4.2.1 Commands for Daily Backups Directly to Tape

Take the following steps to perform daily backups directly to tape:

1. Connect RMAN to the standby database (as target database) and to the recovery manager.
2. Execute the following RMAN commands:

```
RESYNC CATALOG FROM DB_UNIQUE_NAME ALL;
BACKUP AS BACKUPSET INCREMENTAL LEVEL 1 DATABASE PLUS ARCHIVELOG;
DELETE ARCHIVELOG ALL;
```

These commands resynchronize the information from all other databases in the Data Guard environment. They also create a level 1 incremental backup of the database, including all archived logs. On the first day this script is run, if no level 0 backups are found, then a level 0 backup is created.

The `DELETE ARCHIVELOG ALL` command is necessary only if all archived log files are not in a flash recovery area.

11.4.2.2 Commands for Weekly Backups Directly to Tape

One day a week, take the following steps to perform a weekly backup directly to tape:

1. Connect RMAN to the standby database (as target database) and to the recovery catalog.
2. Execute the following RMAN commands:

```
BACKUP AS BACKUPSET INCREMENTAL LEVEL 0 DATABASE PLUS ARCHIVELOG;
DELETE ARCHIVELOG ALL;
```

These commands resynchronize the information from all other databases in the Data Guard environment, and create a level 0 database backup that includes all archived logs.

The `DELETE ARCHIVELOG ALL` command is necessary only if all archived log files are not in a flash recovery area.

11.5 Registering and Unregistering Databases in a Data Guard Environment

Only the primary database must be explicitly registered using the REGISTER DATABASE command. You do this after connecting RMAN to the recovery catalog and primary database as TARGET.

A new standby is automatically registered in the recovery catalog when you connect to a standby database or when the CONFIGURE DB_UNIQUE_NAME command is used to configure the connect identifier.

To unregister information about a specific standby database, you can use the UNREGISTER DB_UNIQUE_NAME command. When a standby database is completely removed from a Data Guard environment, the database information in the recovery catalog can also be removed after you connect to another database in the same Data Guard environment. The backups that were associated with the database that was unregistered are still usable by other databases. You can associate these backups with any other existing database by using the CHANGE BACKUP RESET DB_UNIQUE_NAME command.

When the UNREGISTER DB_UNIQUE_NAME command is used with the INCLUDING BACKUPS option, the metadata for all the backup files associated with the database being removed is also removed from the recovery catalog.

11.6 Reporting in a Data Guard Environment

Use the RMAN LIST, REPORT, and SHOW commands with the FOR DB_UNIQUE_NAME clause to view information about a specific database.

For example, after connecting to the recovery catalog, you could use the following commands to display information for a database with a DBID of 1625818158 and to list the databases in the Data Guard environment. The SET DBID command is required only if you are not connected to a database as TARGET. The last three commands list archive logs, database file names, and RMAN configuration information for a database with a DB_UNIQUE_NAME of BOSTON.

```
SET DBID 1625818158;
LIST DB_UNIQUE_NAME OF DATABASE;
LIST ARCHIVELOG ALL FOR DB_UNIQUE_NAME BOSTON;
REPORT SCHEMA FOR DB_UNIQUE_NAME BOSTON;
SHOW ALL FOR DB_UNIQUE_NAME BOSTON;
```

11.7 Performing Backup Maintenance in a Data Guard Environment

The files in a Data Guard environment (datafiles, archived logs, backup pieces, image copies, and proxy copies) are associated with a database through use of the DB_UNIQUE_NAME parameter. Therefore, it is important that the value supplied for DB_UNIQUE_NAME be unique for each database in a Data Guard environment. This information, along with file-sharing attributes, is used to determine which files can be accessed during various RMAN operations.

File sharing attributes state that files on disk are accessible only at the database with which they are associated, whereas all files on tape are assumed to be accessible by all databases. RMAN commands such as BACKUP and RESTORE, as well as other maintenance commands, work according to this assumption. For example, during a roll-forward operation of an image copy at a database, only image copies associated with the database are rolled forward. Likewise, all incremental backups on disk and all incremental backups on tape will be used to roll forward the image copies. Similarly,

during recovery operations, only disk backups associated with the database and files on tape will be considered as sources for backups.

> **See Also:** *Oracle Database Backup and Recovery Reference* for detailed information about RMAN commands

11.7.1 Changing Metadata in the Recovery Catalog

You can use the RMAN CHANGE command with various operands to change metadata in the recovery catalog, as described in the following sections.

Changing File Association from One Standby Database to Another

Use the CHANGE command with the RESET DB_UNIQUE_NAME option to alter the association of files from one database to another within a Data Guard environment. The CHANGE command is useful when disk backups or archived logs are transferred from one database to another and you want to use them on the database to which they were transferred. The CHANGE command can also change the association of a file from one database to another database, without having to directly connect to either database using the FOR DB_UNIQUE_NAME and RESET DB_UNIQUE_NAME TO options.

Changing DB_UNIQUE_NAME for a Database

If the value of the DB_UNIQUE_NAME initialization parameter changes for a database, the same change must be made in the Data Guard environment. The RMAN recovery catalog, after connecting to that database instance, will know both the old and new value for DB_UNIQUE_NAME. To merge the information for the old and new values within the recovery catalog schema, you must use the RMAN CHANGE DB_UNIQUE_NAME command. If RMAN is not connected to the instance with the changed DB_UNIQUE_NAME parameter, then the CHANGE DB_UNIQUE_NAME command can also be used to rename the DB_UNIQUE_NAME in the recovery catalog schema. For example, if the instance parameter value for a database was changed from BOSTON_A to BOSTON_B, the following command should be executed at the RMAN prompt after connecting to a target database and recovery catalog:

```
CHANGE DB_UNIQUE_NAME FROM BOSTON_A TO BOSTON_B;
```

Making Backups Unavailable or Removing Their Metadata

Use CHANGE command options such as AVAILABLE, UNAVAILABLE, KEEP, and UNCATALOG to make backups available or unavailable for restore and recovery purposes, and to keep or remove their metadata.

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

11.7.2 Deleting Archived Logs or Backups

Use the DELETE command to delete backup sets, image copies, archived logs, or proxy copies. To delete only files that are associated with a specific database, you must use the FOR DB_UNIQUE_NAME option with the DELETE command.

File metadata is deleted for all successfully deleted files associated with the current target database (or for files that are not associated with any known database). If a file could not be successfully deleted, you can use the FORCE option to remove the file's metadata.

When a file associated with another database is deleted successfully, its metadata in the recovery catalog is also deleted. Any files that are associated with other databases, and that could not be successfully deleted, are listed at the completion of the DELETE command, along with instructions for you to perform the same operation at the database with which the files are associated (files are grouped by database). Note that the FORCE option cannot be used to override this behavior. If you are certain that deleting the metadata for the non-deletable files will not cause problems, you can use the CHANGE RESET DB_UNIQUE_NAME command to change the metadata for association of files with the database and use the DELETE command with the FORCE option to delete the metadata for the file.

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

11.7.3 Validating Recovery Catalog Metadata

Use the CROSSCHECK command to validate and update file status in the recovery catalog schema. To validate files associated with a specific database, use the FOR DB_UNIQUE_NAME option with the CROSSCHECK command.

Metadata for all files associated with the current target database (or for any files that are not associated with any database), will be marked AVAILABLE or EXPIRED according to the results of the CROSSCHECK operation.

If a file associated with another database is successfully inspected, its metadata in the recovery catalog is also changed to AVAILABLE. Any files that are associated with other databases, and that could not be inspected successfully, are listed at the completion of the CROSSCHECK command, along with instructions for you to perform the same operation at the database with which the files are associated (files are grouped by site). If you are certain of the configuration and still want to change status metadata for unavailable files, you can use the CHANGE RESET DB_UNIQUE_NAME command to change metadata for association of files with the database and execute the CROSSCHECK command to update status metadata to EXPIRED.

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

11.8 Recovery Scenarios in a Data Guard Environment

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 need to configure the channels for that system before executing restore and recover commands. You can set the configuration for a nonexistent database using the SET DBID command and the CONFIGURE command with FOR DB_UNIQUE_NAME. See the Media Management documentation for more information about how to access RMAN backups from different systems.

The following scenarios are described in this section:

- Recovery from Loss of Datafiles on the Primary Database
- Recovery from Loss of Datafiles on the Standby Database
- Recovery from Loss of a Standby Control File
- Recovery from Loss of the Primary Control File
- Recovery from Loss of an Online Redo Log File
- Incomplete Recovery of the Primary Database

11.8.1 Recovery from Loss of Datafiles on the Primary Database

You can recover from loss of datafiles on the primary database by using backups or by using the files on a standby database, as described in the following sections.

Using Backups

Issue the following RMAN commands to restore and recover datafiles. You must be connected to both the primary and recovery catalog databases.

```
RESTORE DATAFILE n,m...;
RECOVER DATAFILE n,m...;
```

Issue the following RMAN commands to restore and recover tablespaces. You must be connected to both the primary and recovery catalog databases.

```
RESTORE TABLESPACE tbs_name1, tbs_name2, ...
RECOVER TABLESPACE tbs_name1, tbs_name2, ...
```

Using Files On a Standby Database

As of Oracle 11*g*, you can use files on a standby database to recover a lost datafile. This works well if the standby is up-to-date and the network connection is sufficient enough to support the file copy between the standby and primary.

Start RMAN and take the following steps to copy the datafiles from the standby to the primary:

1. Connect to the standby database as the target database:

   ```
   CONNECT TARGET sys@standby
   ```

   You are prompted for a password:

   ```
   target database Password: password
   ```

2. Connect to the primary database as the auxiliary database:

   ```
   CONNECT AUXILIARY sys@primary
   ```

   You are prompted for a password:

   ```
   target database Password: password
   ```

3. Back up the datafile on the *standby* host across the network to a location on the *primary* host. For example, suppose that `/disk1/df2.dbf` is the name of datafile 2 on the standby host. Suppose that `/disk8/datafile2.dbf` is the name of datafile 2 on the primary host. The following command would copy datafile 2 over the network to `/disk9/df2copy.dbf`:

   ```
   BACKUP AS COPY DATAFILE 2 AUXILIARY FORMAT '/disk9/df2copy.dbf';
   ```

4. Exit the RMAN client as follows:

   ```
   EXIT;
   ```

5. Start RMAN and connect to the primary database as target, and to the recovery catalog:

   ```
   CONNECT TARGET sys@primary;
   target database Password: password

   CONNECT CATALOG rman@catdb;
   recovery catalog database Password: password
   ```

6. Use the CATALOG DATAFILECOPY command to catalog this datafile copy so that RMAN can use it.:

```
CATALOG DATAFILECOPY '/disk9/df2copy.dbf';
```

Then use the SWITCH DATAFILE command to switch the datafile copy so that /disk9/df2copy.dbf becomes the current datafile:

```
RUN {
  SET NEWNAME FOR DATAFILE 2 TO '/disk9/df2copy.dbf';
  SWITCH DATAFILE 2;
}
```

11.8.2 Recovery from Loss of Datafiles on the Standby Database

To recover the standby database after the loss of one or more datafiles, you must restore the lost files to the standby database from the backup using the RMAN RESTORE DATAFILE command. If all the archived redo log files required for recovery of damaged files are accessible on disk by the standby database, restart Redo Apply.

If the archived redo log files required for recovery are not accessible on disk, use RMAN to recover the restored datafiles to an SCN/log sequence greater than the last log applied to the standby database, and then restart Redo Apply to continue the application of redo data, as follows:

1. Connect SQL*Plus to the standby database.
2. Stop Redo Apply using the SQL ALTER DATABASE . . . statement.
3. In a separate terminal, start RMAN and connect to both the standby and recovery catalog databases (use the TARGET keyword to connect to the standby instance).
4. Issue the following RMAN commands to restore and recover datafiles on the standby database:

```
RESTORE DATAFILE <n,m,...>;
RECOVER DATABASE;
```

To restore a tablespace, use the RMAN 'RESTORE TABLESPACE *tbs_name1, tbs_name2, ...*' command.

5. At the SQL*Plus prompt, restart Redo Apply using the SQL ALTER DATABASE . . . statement.

> **See Also:** Section 7.3 and Section 7.4 for more information about starting and stopping Redo Apply

11.8.3 Recovery from Loss of a Standby Control File

Oracle software allows multiplexing of the standby control file. To ensure the standby control file is multiplexed, check the CONTROL_FILES initialization parameter, as follows:

```
SQL> SHOW PARAMETER CONTROL_FILES;
NAME                                 TYPE        VALUE
 ------------------------------------ ----------- ------------------------------
control_files                        string      <cfilepath1>,<cfilepath2>
```

If one of the multiplexed standby control files is lost or is not accessible, Oracle software stops the instance and writes the following messages to the alert log:

```
ORA-00210: cannot open the specified controlfile
ORA-00202: controlfile: '/disk1/oracle/dbs/scf3_2.f'
ORA-27041: unable to open file
```

You can copy an intact copy of the control file over the lost copy, then restart the standby instance using the following SQL statements:

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

You can restore the control file from backups by executing the `RESTORE CONTROLFILE` command and then the `RECOVER DATABASE` command. The `RECOVER DATABASE` command automatically fixes the file names in the control file to match the files existing at that database, and recovers the database to the most recently received log sequence at the database.

The other alternative is to create a new control file from the primary database, copy it to all multiplexed locations, and manually rename the data file names to match files existing on disk.

11.8.4 Recovery from 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.

You can restore the control file from backups by executing the `RESTORE CONTROLFILE` command and the `RECOVER DATABASE` command. The `RECOVER DATABASE` command automatically fixes the file names in the control file to match the files existing at that database, and recovers the database.

The other alternative is to create a new control file using `CREATE CONTROLFILE` SQL command. It is possible to re-create the control file provided all data files and online logs are not lost.

> **See Also:** *Oracle Database Backup and Recovery User's Guide* for detailed information about using RMAN to recover from the loss of control files

11.8.5 Recovery from 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: '/disk1/oracle/dbs/t1_log1.f'
ORA-27037: unable to obtain file status
SVR4 Error: 2: No such file or directory
Additional information: 3
```

If the access problem is temporary due to a hardware issue, correct the problem and processing will continue automatically. If the loss is permanent, a new member can be added and the old one dropped from the group.

To add a new member to a redo log group, issue the following statement:

```
SQL> ALTER DATABASE ADD LOGFILE MEMBER 'log_file_name' REUSE TO GROUP n;
```

You can issue this statement even when the database is open, without affecting database availability.

If all members of an inactive group that has been archived are lost, the group can be dropped and re-created.

In all other cases (loss of all online log members for the current `ACTIVE` group, or an inactive group which has not yet been archived), you must fail over to the standby database. Refer to Chapter 8 for the failover procedure.

11.8.6 Incomplete Recovery of the Primary Database

Incomplete recovery of the primary database is normally done in cases such as when the database is logically corrupted (by a 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 primary database. All the procedures are in order of preference, starting with the one that is the least time consuming.

Using Flashback Database Using Flashback Database is the recommended procedure when the Flashback Database feature is enabled on the primary database, none of the database files are lost, and the point-in-time recovery is greater than the oldest flashback SCN or the oldest flashback time. See Section 13.3 for the procedure to use Flashback Database to do point-in-time recovery.

Using the standby database instance This is the recommended procedure when the standby database is behind the desired incomplete recovery time, and Flashback Database is not enabled on the primary or standby databases:

1. Recover the standby database to the desired point in time.

   ```
   RECOVER DATABASE UNTIL TIME 'time';
   ```

   Alternatively, incomplete recovery time can be specified using the SCN or log sequence number:

   ```
   RECOVER DATABASE UNTIL SCN incomplete recovery SCN';
   RECOVER DATABASE UNTIL LOGSEQ incomplete recovery log sequence number THREAD
   thread number;
   ```

2. Open the standby database in read-only mode to verify the state of database.

   If the state is not what is desired, use the LogMiner utility to look at the archived redo log files to find the right target time or SCN for incomplete recovery. Alternatively, you can start by recovering the standby database to a point that you know is before the target time, and then open the database in read-only mode to examine the state of the data. Repeat this process until the state of the database is verified to be correct. Note that if you recover the database too far (that is, past the SCN where the error occurred) you cannot return it to an earlier SCN.

3. Activate the standby database using the SQL `ALTER DATABASE ACTIVATE STANDBY DATABASE` statement. This converts the standby database to a primary database, creates a new resetlogs branch, and opens the database. See Section 9.4 to learn how the standby database reacts to the new reset logs branch.

Using the primary database instance If all of the standby database instances have already been recovered past the desired point in time and Flashback Database is not enabled on the primary or standby database, then this is your only option.

Use the following procedure to perform incomplete recovery on the primary database:

1. Use LogMiner or another means to identify the time or SCN at which all the data in the database is known to be good.
2. Using the time or SCN, issue the following RMAN commands to do incomplete database recovery and open the database with the RESETLOGS option (after connecting to catalog database and primary instance that is in MOUNT state):

```
RUN
{
SET UNTIL TIME 'time';
RESTORE DATABASE;
RECOVER DATABASE;
}
ALTER DATABASE OPEN RESETLOGS;
```

After this process, all standby database instances must be reestablished in the Data Guard configuration.

11.9 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.

11.9.1 Standby Databases Too Geographically Distant to Share Backups

If the standby databases are far apart from one another, the backups taken on them may not be 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 (that is, the flash recovery area does not have to be the same for the primary and standby databases).

In this scenario, you can still use the general strategies described in Section 11.8, 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 *system name* option when creating backups; the RESTORE command must use the FROM TAG *system name* option; and the RECOVER command must use the FROM TAG *system name* ARCHIVELOG TAG *system 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 'system name';
RECOVER DATABASE FROM TAG 'system name' ARCHIVELOG TAG 'system name';
```

4. Restart Redo Apply.

The standby instance will fetch the remaining archived redo log files.

11.9.2 Standby Database Does Not Contain Datafiles, Used as a FAL Server

Use the same procedure described in Section 11.4, 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.

11.9.3 Standby Database File Names Are Different From Primary Database

Note: As of Oracle Database 11*g*, the recovery catalog can resynchronize the file names from each standby database site. However, if the file names from a standby database were never resynchronized for some reason, then you can use the procedure described in this section to do so.

If the database filenames are not the same on the primary and standby databases that were never resynchronized, the RESTORE and RECOVER commands you use will be slightly different. To obtain the actual datafile names on the standby database, query the V$DATAFILE view and specify the SET NEWNAME option for all the datafiles in the database:

```
RUN
{
SET NEWNAME FOR DATAFILE 1 TO 'existing file location for file#1 from V$DATAFILE';
SET NEWNAME FOR DATAFILE 2 TO 'existing file location for file#2 from V$DATAFILE';
…
…
 SET NEWNAME FOR DATAFILE n TO 'existing file location for file#n from V$DATAFILE';
 RESTORE {DATAFILE <n,m,…> | TABLESPACE tbs_name_1, 2, …| DATABASE;
SWITCH DATAFILE ALL;
RECOVER DATABASE {NOREDO};
}
```

Similarly, the RMAN DUPLICATE command should also use the SET NEWNAME option to specify new filenames during standby database creation. Or you could set the LOG_FILE_NAME_CONVERT and DB_FILE_NAME_CONVERT parameters.

11.10 Using RMAN Incremental Backups to Roll Forward a Physical Standby Database

In some situations, RMAN incremental backups can be used to synchronize a physical standby database with the primary database. You can use the RMAN BACKUP INCREMENTAL FROM SCN command to create a backup on the primary database that starts at the current SCN of the standby, which can then be used to roll the standby database forward in time.

The steps described in this section apply to situations in which RMAN incremental backups may be useful because the physical standby database either:

- Lags far behind the primary database

- Has widespread nologging changes
- Has nologging changes on a subset of datafiles

Note: Oracle recommends the use of a recovery catalog when performing this operation. These steps are possible without a recovery catalog, but great care must be taken to correct the file names in the restored control file.

See Also: *Oracle Database Backup and Recovery User's Guide* for more information about RMAN incremental backups

11.10.1 Steps for Using RMAN Incremental Backups

Except where stated otherwise, the following steps apply to all three situations just listed.

1. Stop Redo Apply on the standby database:

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

2. On the standby database, compute the `FROM SCN` for the incremental backup. This is done differently depending on the situation:

   - On a standby that lags far behind the primary database, query the `V$DATABASE` view and record the current SCN of the standby database:

```
SQL> SELECT CURRENT_SCN FROM V$DATABASE;
CURRENT_SCN
-----------
     233995
```

   - On a standby that has widespread nologging changes, query the `V$DATAFILE` view to record the lowest `FIRST_NONLOGGED_SCN`:

```
SQL> SELECT MIN(FIRST_NONLOGGED_SCN) FROM V$DATAFILE
  2> WHERE FIRST_NONLOGGED_SCN>0;

MIN(FIRST_NONLOGGED_SCN)
------------------------
                  223948
```

   - On a standby that has nologging changes on a subset of datafiles, query the `V$DATAFILE` view, as follows:

```
SQL> SELECT FILE#, FIRST_NONLOGGED_SCN FROM V$DATAFILE
  2> WHERE FIRST_NONLOGGED_SCN > 0;

FILE#      FIRST_NONLOGGED_SCN
---------- -------------------
         4              225979
         5              230184
```

3. Connect to the primary database as the RMAN target and create an incremental backup from the current SCN (for a standby lagging far behind the primary) or the lowest `FIRST_NONLOGGED_SCN` (for a standby with widespread nologging changes) of the standby database that was recorded in step 2:

```
RMAN> BACKUP INCREMENTAL FROM SCN 233995 DATABASE FORMAT '/tmp/ForStandby_%U'
tag 'FORSTANDBY';
```

If the standby has nologging changes on a subset of datafiles, then create an incremental backup for each datafile listed in the FIRST_NONLOGGED_SCN column (recorded in step 1), as follows:

```
RMAN> BACKUP INCREMENTAL FROM SCN 225979 DATAFILE 4 FORMAT '/tmp/ForStandby_%U'
TAG 'FORSTANDBY';
RMAN> BACKUP INCREMENTAL FROM SCN 230184 DATAFILE 5 FORMAT '/tmp/ForStandby_%U'
TAG 'FORSTANDBY';
```

4. If backups were written to shared storage, skip this step. Otherwise, transfer all backup sets created on the primary system to the standby system and then catalog them. There may have been more than one backup file created. The following example, entered at the operating system prompt, uses the scp command to copy the files:

   ```
   scp /tmp/ForStandby_* standby:/tmp
   ```

   Then, at the RMAN prompt, enter the following command to catalog them:

   ```
   RMAN> CATALOG START WITH '/tmp/ForStandby';
   ```

5. Connect to the standby database as the RMAN target and execute the REPORT SCHEMA statement to ensure that the standby database site is automatically registered and that the files names at the standby site are displayed:

   ```
   RMAN> REPORT SCHEMA;
   ```

6. Connect to the standby database as the RMAN target and apply incremental backups. Do the following:

   ```
   RMAN> STARTUP FORCE NOMOUNT;
   RMAN> RESTORE STANDBY CONTROLFILE FROM TAG 'FORSTANDBY';
   RMAN> ALTER DATABASE MOUNT;
   RMAN> RECOVER DATABASE NOREDO;
   ```

   Note: Oracle recommends that you use a recovery catalog, but if you do not, then just prior to issuing the RECOVER command, you must edit the file names in your control file or use the RMAN SET NEWNAME command to assign the datafile names.

7. On standbys that have widespread nologging changes or that have nologging changes on a subset of datafiles, query the V$DATAFILE view to verify there are no datafiles with nologged changes. The following query should return zero rows:

   ```
   SQL> SELECT FILE#, FIRST_NONLOGGED_SCN FROM V$DATAFILE
     2> WHERE FIRST_NONLOGGED_SCN > 0;
   ```

   Note: The incremental backup will become obsolete in 7 days, or you can remove it now using the RMAN DELETE command.

8. Re-create the standby control file.
9. Start Redo Apply on the physical standby database:

   ```
   SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE
     2> USING CURRENT LOGFILE DISCONNECT FROM SESSION;
   ```

12

Using SQL Apply to Upgrade the Oracle Database

Starting with Oracle Database 10*g* release 1 (10.1.0.3), you can use a logical standby database to perform a *rolling upgrade* of Oracle Database 10*g* software. During a rolling upgrade, you can run different releases of an Oracle database on the primary and logical standby databases while you upgrade them, one at a time, incurring minimal downtime on the primary database.

> **Note:** This chapter describes an alternative to the usual upgrade procedure involving longer downtime, as described in Appendix B, "Upgrading Databases in a Data Guard Configuration". Do not attempt to combine steps from the method described in this chapter with steps from Appendix B.

The instructions in this chapter describe how to minimize downtime while upgrading an Oracle database. This chapter provides the following topics:

- Benefits of a Rolling Upgrade Using SQL Apply
- Requirements to Perform a Rolling Upgrade Using SQL Apply
- Figures and Conventions Used in the Upgrade Instructions
- Performing a Rolling Upgrade By Creating a New Logical Standby Database
- Performing a Rolling Upgrade With an Existing Logical Standby Database
- Performing a Rolling Upgrade With an Existing Physical Standby Database

12.1 Benefits of a Rolling Upgrade Using SQL Apply

Performing a rolling upgrade with SQL Apply provides several advantages:

- Your database will incur very little downtime. The overall downtime can be as little as the time it takes to perform a switchover.
- You eliminate application downtime due to PL/SQL recompilation.
- You can validate the upgraded database release without affecting the primary database.

12.2 Requirements to Perform a Rolling Upgrade Using SQL Apply

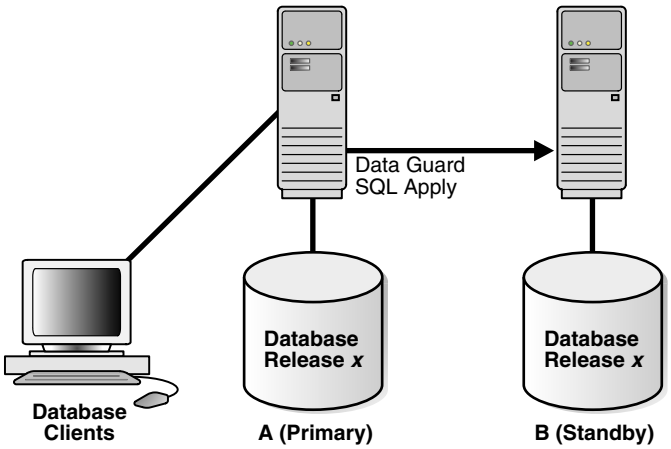
The rolling upgrade procedure requires the following:

- A primary database that is running Oracle Database release *x* and a logical standby database that is running Oracle Database release *y*.
- The databases must not be part of a Data Guard Broker configuration. See *Oracle Data Guard Broker* for information about removing databases from a broker configuration.
- The Data Guard protection mode must be set to either maximum availability or maximum performance. Query the `PROTECTION_LEVEL` column in the `V$DATABASE` view to find out the current protection mode setting.
- To ensure the primary database can proceed while the logical standby database is being upgraded, the `LOG_ARCHIVE_DEST_`*n* initialization parameter for the logical standby database destination must *not* be set to `MANDATORY`.
- The `COMPATIBLE` initialization parameter must match the software release prior to the upgrade. That is, a rolling upgrade from release *x* to release *y* requires that the `COMPATIBLE` initialization parameter be set to release *x* on both the primary and standby databases.

12.3 Figures and Conventions Used in the Upgrade Instructions

Figure 12–1 shows a Data Guard configuration before the upgrade begins, with the primary and logical standby databases both running the same Oracle Database software release.

Figure 12–1 Data Guard Configuration Before Upgrade



During the upgrade process, the Data Guard configuration operates with mixed database releases at several points in this process. Data protection is not available across releases. During these steps, consider having a second standby database in the Data Guard configuration to provide data protection.

The steps and figures describing the upgrade procedure refer to the databases as "Database A" and "Database B" rather than as the "primary database" and "standby database." This is because the databases switch roles during the upgrade procedure. Initially, Database A is the primary database and Database B is the logical standby database, as shown in Figure 12–1.

The following sections describe scenarios in which you can use the SQL Apply rolling upgrade procedure:

12.4 Performing a Rolling Upgrade By Creating a New Logical Standby Database

This scenario assumes that you do not have an existing Data Guard configuration, but you are going to create a logical standby database solely for the purpose of performing a rolling upgrade of the Oracle Database.

Perform the following steps to prepare the primary and standby databases for upgrading.

Step 1 Identify unsupported data types and storage attributes

To identify unsupported database objects on the primary database and decide how to handle them, follow these steps:

1. Identify unsupported data types and storage attributes for tables:
   - Review the list of supported data types and storage attributes provided in Appendix C, "Data Type and DDL Support on a Logical Standby Database".
   - Query the `DBA_LOGSTDBY_UNSUPPORTED` and `DBA_LOGSTDBY_SKIP` views on the primary database. Changes that are made to the listed tables and schemas on the primary database will not be applied on the logical standby database. The following query shows an example of a list of unsupported tables:

```
SQL> SELECT DISTINCT OWNER, TABLE_NAME FROM DBA_LOGSTDBY_UNSUPPORTED;
OWNER      TABLE_NAME
---------- -----------------
OE         CATEGORIES_TAB
OE         CUSTOMERS
OE         WAREHOUSES
PM         ONLINE_MEDIA
PM         PRINT_MEDIA
SCOTT      MYCOMPRESS
SH         MVIEW$_EXCEPTIONS
7 rows selected.

SQL>
SQL> SELECT OWNER FROM DBA_LOGSTDBY_SKIP
  2   WHERE STATEMENT_OPT = 'INTERNAL SCHEMA';

OWNER
------------------------------
CTXSYS
DBSNMP
DIP
ORDPLUGINS
ORDSYS
OUTLN
SI_INFORMTN_SCHEMA
SYS
```

```
SYSTEM
WMSYS
10 rows selected.
```

2. Decide how to handle unsupported tables.

   If unsupported objects are being modified on your primary database, it might be possible to perform the upgrade anyway by temporarily suspending changes to the unsupported tables for the period of time it takes to perform the upgrade procedure.

   If you can prevent changes to unsupported changes, then using SQL Apply might still be a viable way to perform the upgrade procedure. This method requires that you prevent users from modifying any unsupported tables from the time you create the logical standby control file to the time you complete the upgrade. For example, assume that the Payroll department updates an object table, but that department updates the database only Monday through Friday. However, the Customer Service department requires database access 24 hours a day, 7 days a week, but uses only supported data types and tables. In this scenario, you could perform the upgrade over a weekend. You can monitor transaction activity in the `DBA_LOGSTDBY_EVENTS` view and discontinue the upgrade (if necessary) up to the time you perform the first switchover.

   If you cannot prevent changes to unsupported tables during the upgrade, any unsupported transactions that occur are recorded in the `DBA_LOGSTDBY_EVENTS` table on the logical standby database. After the upgrade is completed, you might be able to use Oracle Data Pump or the Export/Import utility to import the changed tables to the upgraded databases.

   The size of the changed tables will determine how long database operations will be unavailable, so you must decide if a table is too large to export and import its data into the standby database. For example, a 4-terabyte table is not a good candidate for the export/import process.

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

Step 2 Create a logical standby database

To create a logical standby database, follow the instructions in Chapter 4.

Oracle recommends configuring a standby redo log on the logical standby database to minimize downtime.

Step 3 Perform a rolling upgrade

Now that you have created a logical standby database, you can follow the procedure described in Section 12.5, "Performing a Rolling Upgrade With an Existing Logical Standby Database", which assumes that you have a logical standby running the same Oracle software.

12.5 Performing a Rolling Upgrade With an Existing Logical Standby Database

This section provides a step-by-step procedure for upgrading the logical standby database and the primary database. Table 12–1 lists the steps.

Table 12–1 Steps to Perform a Rolling Upgrade WIth an Existing Logical Standby

Step	Description
1	Prepare for rolling upgrade
2	Upgrade the logical standby database
3	Obtain information about unsupported tables
4	Restart SQL Apply on the upgraded logical standby database
5	Monitor events on the upgraded standby database
6	Begin a switchover
7	Import any tables that were modified during the upgrade
8	Complete the switchover and activate user applications
9	Upgrade the old primary database
10	Start SQL Apply on the old primary database
11	Optionally, raise the compatibility level on both databases
12	Monitor events on the new logical standby database
13	Optionally, perform another switchover

Step 1 Prepare for rolling upgrade

Follow these steps to prepare to perform a rolling upgrade of Oracle Software:

1. Stop SQL Apply by issuing the following statement on the logical standby database (Database B):

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

2. Set compatibility, if needed, to the highest value.

   Ensure the `COMPATIBLE` initialization parameter specifies the release number for the Oracle Database software running on the primary database prior to the upgrade.

   For example, if the primary database is running release 10.1, then set the `COMPATIBLE` initialization parameter to 10.1 on both databases. Be sure to set the `COMPATIBLE` initialization parameter on the standby database first *before* you set it on the primary database.

Step 2 Upgrade the logical standby database

Upgrade Oracle database software on the logical standby database (Database B) to release *y*. While the logical standby database is being upgraded, it will not accept redo data from the primary database.

Step 3 Obtain information about unsupported tables

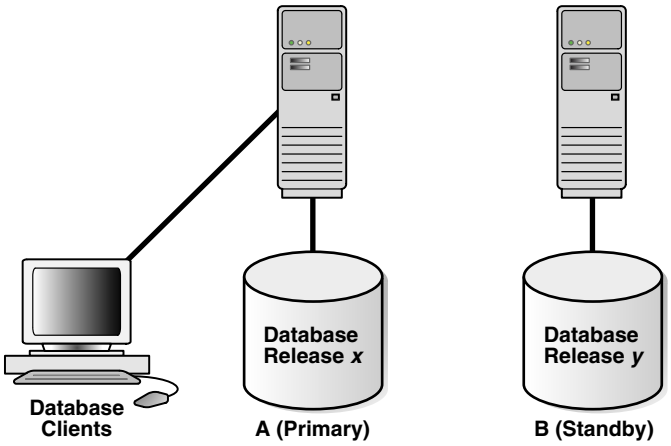
Oracle recommends that you use the `DBMS_LOGSTDBY` PL/SQL procedure on Database B to capture information about transactions running on the primary database that will not be supported by a logical standby database. Run the following procedures to capture and record the information as events in the `DBA_LOGSTDBY_EVENTS` table:

```
EXEC DBMS_LOGSTDBY.APPLY_SET('MAX_EVENTS_RECORDED',DBMS_LOGSTDBY.MAX_EVENTS);
EXEC DBMS_LOGSTDBY.APPLY_SET('RECORD_UNSUPPORTED_OPERATIONS', 'TRUE');
```

To upgrade Oracle Database software, refer to the *Oracle Database Upgrade Guide* for the applicable Oracle Database release.

Figure 12–2 shows Database A running release *x*, and Database B running release *y*. During the upgrade, redo data accumulates on the primary system.

Figure 12–2 Upgrade the Logical Standby Database Release



See Also:

- Section 10.5.1, "Customizing Logging of Events in the DBA_LOGSTDBY_EVENTS View" on page 10-16 for more information about the `DBA_LOGSTDBY_EVENTS` view
- *Oracle Database PL/SQL Packages and Types Reference* for complete information about the `DBMS_LOGSTDBY` package

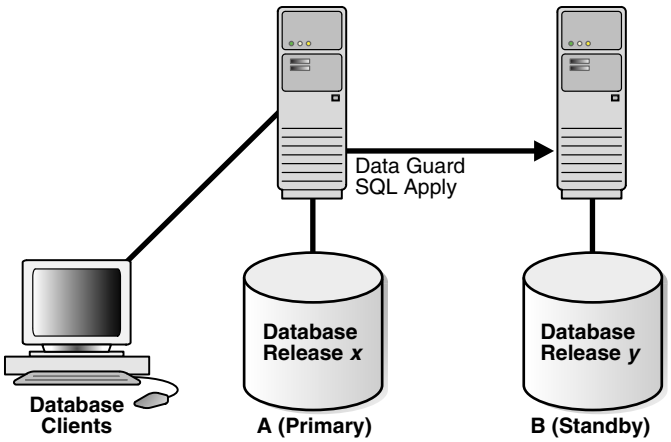
Step 4 Restart SQL Apply on the upgraded logical standby database

Restart SQL Apply and operate with release *x* on Database A and release *y* on Database B. To start SQL Apply, issue the following statement on Database B:

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

The redo data that was accumulating on the primary system is automatically transmitted and applied on the newly upgraded logical standby database. The Data Guard configuration can run the mixed releases shown in Figure 12–3 for an arbitrary period while you verify that the upgraded Oracle Database software release is running properly in the production environment.

Figure 12–3 Running Mixed Releases



To monitor how quickly Database B is catching up to Database A, query the V$LOGSTDBY_PROGRESS view on Database B. For example:

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

SQL> SELECT SYSDATE, APPLIED_TIME FROM V$LOGSTDBY_PROGRESS;

SYSDATE            APPLIED_TIME
------------------ ------------------
27-JUN-05 17:07:06 27-JUN-05 17:06:50
```

Step 5 Monitor events on the upgraded standby database

You should frequently query the DBA_LOGSTDBY_EVENTS view to learn if there are any DDL and DML statements that have not been applied on Database B. Example 12–1 demonstrates how monitoring events can alert you to potential differences in the two databases.

Example 12–1 Monitoring Events with DBA_LOGSTDBY_EVENTS

```
SQL> SET LONG 1000
SQL> SET PAGESIZE 180
SQL> SET LINESIZE 79
SQL> SELECT EVENT_TIMESTAMP, EVENT, STATUS FROM DBA_LOGSTDBY_EVENTS
ORDER BY EVENT_TIMESTAMP;

EVENT_TIMESTAMP
---------------------------------------------------------------------------
EVENT
---------------------------------------------------------------------------
STATUS
---------------------------------------------------------------------------
…
24-MAY-05 05.18.29.318912 PM
CREATE TABLE SYSTEM.TST (one number)
ORA-16226: DDL skipped due to lack of support

24-MAY-05 05.18.29.379990 PM
"SYSTEM"."TST"
ORA-16129: unsupported dml encountered
```

In the preceding example:

- The `ORA-16226` error shows a DDL statement that could not be supported. In this case, it could not be supported because it belongs to an internal schema.
- The `ORA-16129` error shows that a DML statement was not applied.

These types of errors indicate that not all of the changes that occurred on Database A have been applied to Database B. At this point, you must decide whether or not to continue with the upgrade procedure. If you are certain that this difference between the logical standby database and the primary database is acceptable, then continue with the upgrade procedure. If not, discontinue and reinstantiate Database B and perform the upgrade procedure at another time.

Step 6 Begin a switchover

When you are satisfied that the upgraded database software is operating properly, perform a switchover to reverse the database roles by issuing the following statement on Database A:

```
SQL> ALTER DATABASE COMMIT TO SWITCHOVER TO LOGICAL STANDBY;
```

This statement must wait for existing transactions to complete. To minimize the time it takes to complete the switchover, users still connected to Database A should log off immediately and reconnect to Database B.

Note: The usual two-phased prepared switchover described in 7.3.1 cannot be used because it requires both primary and standby databases to be running the same version of the Oracle software and at this point, the primary database is running a lower version of the Oracle software. Instead, the single-phased unprepared switchover procedure documented above is used. The unprepared switchover should only be used in the context of a rolling upgrade using logical standby database.

Note: If you suspended activity to unsupported tables or packages on Database A when it was the primary database, you must continue to suspend the same activities on Database B while it is the primary database if you eventually plan to switch back to Database A.

Step 7 Import any tables that were modified during the upgrade

Step 5 "Monitor events on the upgraded standby database" described how to list unsupported tables that are being modified. If unsupported DML statements were issued on the primary database (as described in Example 12–1), import the latest version of those tables using an import utility such as Oracle Data Pump.

For example, the following import command truncates the `scott.emp` table and populates it with data matching the former primary database (A):

```
IMPDP SYSTEM NETWORK_LINK=DATABASEA TABLES=SCOTT.EMP TABLE_EXIST_ACTION=TRUNCATE
```

Note that this command will prompt you for the `impdp` password before executing.

Step 8 Complete the switchover and activate user applications

When you are satisfied that the upgraded database software is operating properly, complete the switchover to reverse the database roles:

1. On Database B, query the SWITCHOVER_STATUS column of the V$DATABASE view, as follows:

```
SQL> SELECT SWITCHOVER_STATUS FROM V$DATABASE;

SWITCHOVER_STATUS
--------------------
TO PRIMARY
```

2. When the SWITCHOVER_STATUS column displays TO PRIMARY, complete the switchover by issuing the following statement on Database B:

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

> **Note:** The usual two-phased prepared switchover described in 7.3.1 cannot be used because it requires both primary and standby databases to be running the same version of the Oracle software and at this point, the primary database is running a lower version of the Oracle software. Instead, the single-phased unprepared switchover procedure documented above is used. The unprepared switchover should only be used in the context of a rolling upgrade using logical standby database.

3. Activate the user applications and services on Database B, which is now running in the primary database role.

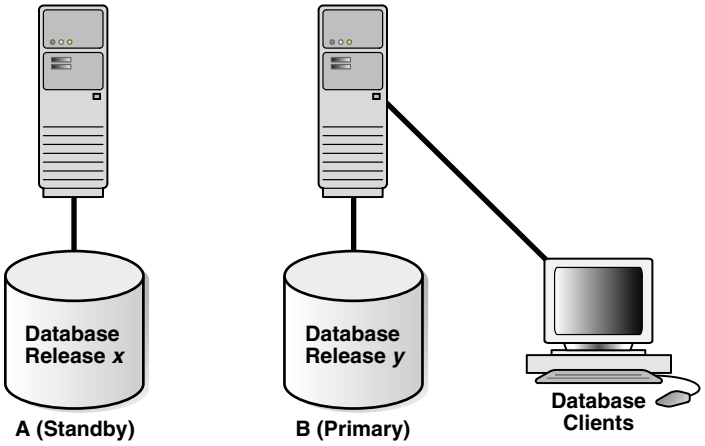
After the switchover, you cannot send redo data from the new primary database (B) that is running the new database software release to the new standby database (A) that is running an older software release. This means the following:

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

Figure 12–4 shows Database B, the former standby database (running release *y*), is now the primary database, and Database A, the former primary database (running release *x*), is now the standby database. The users are connected to Database B.

If Database B can adequately serve as the primary database and your business does not require a logical standby database to support the primary database, then you have completed the rolling upgrade process. Allow users to log in to Database B and begin working there, and discard Database A when it is convenient. Otherwise, continue with step 9.

Figure 12–4 After a Switchover



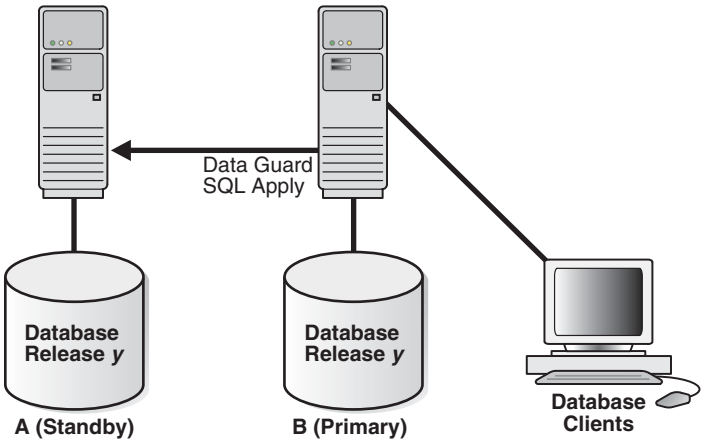
Step 9 Upgrade the old primary database

Database A is still running release *x* and cannot apply redo data from Database B until you upgrade it and start SQL Apply.

For more information about upgrading Oracle Database software, see the *Oracle Database Upgrade Guide* for the applicable Oracle Database release.

Figure 12–5 shows the system after both databases have been upgraded.

Figure 12–5 Both Databases Upgraded



Step 10 Start SQL Apply on the old primary database

Issue the following statement to start SQL Apply on Database A and, if necessary, create a database link to Database B:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE NEW PRIMARY db_link_to_b;
```

Note: You will need to create a database link (if one has not already been set up) and to use the `NEW PRIMARY` clause, because in Step 4 the single-phased unprepared switchover was used to turn Database A into a standby database.

You will need to connect as `SYS` user or with an account with similar level of privileges for the database link.

When you start SQL Apply on Database A, the redo data that is accumulating on the primary database (B) is sent to the logical standby database (A). The primary database is protected against data loss once all the redo data is available on the standby database.

Step 11 Optionally, raise the compatibility level on both databases

Raise the compatibility level of both databases by setting the `COMPATIBLE` initialization parameter. You must raise the compatibility level at the logical standby database before you raise it at the primary database. Set the `COMPATIBLE` parameter on the standby database before you set it on the primary database. See *Oracle Database Reference* for more information about the `COMPATIBLE` initialization parameter.

Step 12 Monitor events on the new logical standby database

To ensure that all changes performed on Database B are properly applied to the logical standby database (A), you should frequently query the `DBA_LOGSTDBY_EVENTS` view, as you did for Database A in step 5. (See Example 12–1.)

If changes were made that invalidate Database A as a copy of your existing primary database, you can discard Database A and create a new logical standby database in its place. See Chapter 4, "Creating a Logical Standby Database" for complete information.

Step 13 Optionally, perform another switchover

Optionally, perform another switchover of the databases so Database A is once again running in the primary database role (as shown in Figure 12–1).

Note: You will use the two-phased prepared switchover described in 7.3.1 since at this time, both Database A and Database B are running the same version of the Oracle software.

See Also: Section 8.3.1, "Performing a Switchover to a Logical Standby Database"

12.6 Performing a Rolling Upgrade With an Existing Physical Standby Database

The steps in this section show you how to perform a rolling upgrade of Oracle software and then get back to your original configuration in which A is the primary database and B is the physical standby database, and both of them are running the upgraded Oracle software.

Note: The steps in this section assume that you have a primary database (A) and a physical standby database (B) already set up and using Oracle Database release 11.1 or later.

Table 12–2 summarizes the steps involved.

Table 12–2 Steps to Perform a Rolling Upgrade With an Existing Physical Standby

Step	Description
1	Prepare the primary database for a rolling upgrade (perform these steps on Database A)
2	Convert the physical standby database into a logical standby database (perform these steps on Database B)
3	Upgrade the logical standby database and catch up with the primary database (perform these steps on Database B)
4	Flashback Database A to the guaranteed restore point (perform these steps on Database A)
5	Bring up Database A as a logical standby to Database B using the new version of Oracle software
6	Convert Database A back to a physical standby
7	Start managed recovery on Database A
8	Perform a switchover to make Database A the primary database

Step 1 Prepare the primary database for a rolling upgrade (perform these steps on Database A)

1. Enable Flashback Database, if it is not already enabled:

```
SQL> SHUTDOWN IMMEDIATE;
SQL> STARTUP MOUNT;
SQL> ALTER DATABASE FLASHBACK ON;
SQL> ALTER DATABASE OPEN;
```

2. Create a guaranteed restore point:

```
SQL> CREATE RESTORE POINT pre_upgrade GUARANTEE FLASHBACK DATABASE;
```

Step 2 Convert the physical standby database into a logical standby database (perform these steps on Database B)

1. Follow the steps outlined in Chapter 4, "Creating a Logical Standby Database" except for the following difference. In Section 4.2.4.1, "Convert to a Logical Standby Database" you must use a different command to convert the logical standby database. Instead of `ALTER DATABASE RECOVER TO LOGICAL STANDBY db_name`, issue the following command:

```
SQL> ALTER DATABASE RECOVER TO LOGICAL STANDBY KEEP IDENTITY;
SQL> ALTER DATABASE OPEN;
```

2. Disable automatic deletion of foreign archived logs at the logical standby database and then start SQL Apply for the first time:

```
SQL> execute DBMS_LOGSTDBY.APPLY_SET('LOG_AUTO_DELETE', 'FALSE');
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE;
```

Note: You should not delete any remote archived logs processed by the logical standby database (Database B). These remote archived logs are required later during the rolling upgrade process. If you are using the recovery area to store the remote archived logs, you must ensure that it has enough space to accommodate these logs without interfering with the normal operation of the logical standby database.

Step 3 Upgrade the logical standby database and catch up with the primary database (perform these steps on Database B)

You can now follow Steps 1 through 6 as described in Section 12.5, "Performing a Rolling Upgrade With an Existing Logical Standby Database". At the end of these steps, Database B will be your primary database running the upgraded version of the Oracle software, and Database A has become your logical standby database.

Move on to the next step to turn Database A into the physical standby for Database B.

Step 4 Flashback Database A to the guaranteed restore point (perform these steps on Database A)

```
SQL> SHUTDOWN IMMEDIATE;
SQL> STARTUP MOUNT;
SQL> FLASHBACK DATABASE TO RESTORE POINT pre_upgrade;
SQL> SHUTDOWN IMMEDIATE;
```

Step 5 Bring up Database A as a logical standby to Database B using the new version of Oracle software

At this point, you should switch the Oracle binary at Database A to use the higher version of the Oracle software. You will not run the upgrade scripts, since Database A will be turned into a physical standby, and will be upgraded automatically as it applies the redo data generated by Database B.

Mount Database A, as follows:

```
SQL> STARTUP MOUNT;
```

Step 6 Convert Database A back to a physical standby

```
SQL> ALTER DATABASE CONVERT TO PHYSICAL STANDBY;
SQL> SHUTDOWN IMMEDIATE;
```

Step 7 Start managed recovery on Database A

Database A will be upgraded automatically as it applies the redo data generated by Database B. Managed recovery will wait until the new incarnation branch from the primary is registered before it starts applying redo.

```
SQL> STARTUP MOUNT;
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE USING CURRENT LOGFILE
DISCONNECT FROM SESSION;
```

Step 8 Perform a switchover to make Database A the primary database

At this point, Database B is your primary database and Database A is your physical standby, both running the higher version of the Oracle software. To make Database A the primary database, follow the steps described in Section 8.2.1, "Performing a Switchover to a Physical Standby Database" on page 8-7.

13 Data Guard Scenarios

This chapter describes scenarios you might encounter while administering your Data Guard configuration. Each scenario can be adapted to your specific environment. Table 13–1 lists the scenarios presented in this chapter.

Table 13–1 Data Guard Scenarios

Reference	Scenario
Section 13.1	Configuring Logical Standby Databases After a Failover
Section 13.2	Converting a Failed Primary Into a Standby Database Using Flashback Database
Section 13.3	Using Flashback Database After Issuing an Open Resetlogs Statement
Section 13.4	Recovering After the NOLOGGING Clause Is Specified
Section 13.5	Creating a Standby Database That Uses OMF or ASM
Section 13.6	Recovering From Lost-Write Errors on a Primary Database
Section 13.7	Converting a Failed Primary into a Standby Database Using RMAN Backups

13.1 Configuring Logical Standby Databases After a Failover

This section presents the steps required on a logical standby database after the primary database has failed over to another standby database. After a failover has occurred, a logical standby database cannot act as a standby database for the new primary database until it has applied the final redo from the original primary database. This is similar to the way the new primary database applied the final redo during the failover. The steps you must perform depend on whether the new primary database was a physical standby or a logical standby database prior to the failover:

- Section 13.1.1, "When the New Primary Database Was Formerly a Physical Standby Database"
- Section 13.1.2, "When the New Primary Database Was Formerly a Logical Standby Database"

13.1.1 When the New Primary Database Was Formerly a Physical Standby Database

This scenario demonstrates how to configure a logical standby database to support a new primary database that was a physical standby database before it assumed the primary role. In this scenario, SAT is the logical standby database and NYC is the primary database.

Step 1 Disable archiving from the primary database.

On the NYC database, issue the following statements (assuming LOG_ARCHIVE_DEST_4 is configured to archive to the SAT database):

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_4=DEFER;
SQL> ALTER SYSTEM ARCHIVE LOG CURRENT;
```

Step 2 Verify the logical standby database is capable of serving as a standby database to the new primary database.

On the SAT database, issue the following statement:

```
SQL> EXECUTE DBMS_LOGSTDBY.PREPARE_FOR_NEW_PRIMARY(-
     former_standby_type => 'PHYSICAL' -
     dblink => 'nyc_link');
```

Note: If the ORA-16109 message is returned and the 'LOGSTDBY: prepare_for_new_primary failure -- applied too far, flashback required.' warning is written in the alert.log, perform the following steps:

1. Flash back the database to the SCN as stated in the warning and then
2. Repeat this step before continuing.

See Section 13.2.3 for an example of how to flash back a logical standby database to an Apply SCN.

Step 3 Enable archiving on the primary database.

On the NYC database, issue the following statements (assume LOG_ARCHIVE_DEST_4 is configured to archive to the SAT database):

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_4=ENABLE;
SQL> ALTER SYSTEM ARCHIVE LOG CURRENT;
```

Step 4 Query the new primary database to determine the SCN at which real-time apply can be enabled on the logical standby database

On the NYC database, issue the following query to determine the SCN of interest:

```
SQL> SELECT MAX(NEXT_CHANGE#) -1 AS WAIT_FOR_SCN FROM V$ARCHIVED_LOG;
```

Step 5 Start SQL Apply.

On the SAT database, issue the following statement:

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

Note that you must always issue this statement without the real-time apply option. You need to wait for SQL Apply to apply past WAIT_FOR_SCN returned in Step 4, before you can enable real-time apply. To determine when it is safe to resume real-time apply on the logical standby database, monitor the V$LOGSTDBY_PROGRESS view:

```
SQL> SELECT APPLIED_SCN FROM V$LOGSTDBY_PROGRESS;
```

When the value returned is greater than or equal to the WAIT_FOR_SCN value returned in Step 4, you can stop SQL Apply and restart it with real-time apply option:

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

13.1.2 When the New Primary Database Was Formerly a Logical Standby Database

This scenario demonstrates how to configure a logical standby database to support a new primary database that was a logical standby database before it assumed the primary role. In this scenario, SAT is the logical standby database and NYC is the primary database.

Step 1 Ensure the new primary database is ready to support logical standby databases.

On the NYC database, ensure the following query returns a value of `READY`. Otherwise, the LSP1 background process has not completed its work and the configuration of this logical must wait. For example:

```
SQL> SELECT VALUE FROM SYSTEM.LOGSTDBY$PARAMETERS
2>   WHERE NAME = 'REINSTATEMENT_STATUS';
```

> **Note:** If the `VALUE` column contains `NOT POSSIBLE` it means that no logical standby database may be configured with the new primary database, and you must reinstate the database.

Step 2 Disable archiving from the primary database.

On the NYC database, issue the following statements (assume `LOG_ARCHIVE_DEST_4` is configured to archive to the SAT database):

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_4=DEFER;
SQL> ALTER SYSTEM ARCHIVE LOG CURRENT;
```

Step 3 Verify the logical standby database is capable of being a standby to the new primary.

On the SAT database, issue the following statement:

```
SQL> EXECUTE DBMS_LOGSTDBY.PREPARE_FOR_NEW_PRIMARY(-
     former_standby_type => 'LOGICAL' -
     dblink => 'nyc_link');
```

> **Note:** If the `ORA-16109` message is returned and the 'LOGSTDBY: prepare_for_new_primary failure -- applied too far, flashback required.' warning is written in the alert.log file, perform the following steps:
>
> 1. Flash back the database to the SCN as stated in the warning and then
> 2. Repeat this step before continuing.
>
> See Section 13.2.3 for an example of how to flash back a logical standby database to an Apply SCN.

Step 4 Determine the log files that must be copied to the local system.

On the SAT database, look for the output from the `DBMS_LOGSTDBY.PREPARE_FOR_NEW_PRIMARY` procedure that identifies the log files that must be copied to the local system. If Step 3 identified the failover as a *no-data-loss* failover, then the displayed log files must be copied from the new primary database and should not be obtained from other logical standby databases or the former primary database. For example, on a Linux system, you would enter the `grep` command:

```
%grep 'LOGSTDBY: Terminal log' alert_sat.log
LOGSTDBY: Terminal log: [/oracle/dbs/hq_nyc_13.log]
```

> **Note:** If the prior step was executed multiple times, the output from the most recent attempt is the only relevant output. File paths are relative to the new primary database and may not be resolvable on the local file system.

Step 5 Copy the log files to the local system.

On the SAT database, copy the terminal log files to the local system. The following example shows how to do this using Linux commands:

```
%cp /net/nyc/oracle/dbs/hq_nyc_13.log
/net/sat/oracle/dbs/hq_sat_13.log
```

Step 6 Register the terminal log with logical standby database.

On the SAT database, issue the following statement:

```
SQL> ALTER DATABASE REGISTER OR REPLACE LOGICAL LOGFILE -
     '/net/sat/oracle/dbs/hq_sat_13.log';
```

Step 7 Start SQL Apply.

On the SAT database, issue the following statements:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY NEW PRIMARY nyc_link;
```

Note that you must always issue this statement without the real-time apply option. If you want to enable real-time apply on the logical standby database, wait for the above statement to complete successfully, and then issue the following statements:

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

Step 8 Enable archiving on the primary database to the logical standby database.

On the NYC database, issue the following statements (assuming `LOG_ARCHIVE_DEST_4` is configured to archive to the SAT database):

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_4=ENABLE;
SQL> ALTER SYSTEM ARCHIVE LOG CURRENT;
```

13.2 Converting a Failed Primary Into a Standby Database Using Flashback Database

After a failover occurs, the original primary database can no longer participate in the Data Guard configuration until it is repaired and established as a standby database in the new configuration. To do this, you can use the Flashback Database feature to recover the failed primary database to a point in time before the failover occurred, and then convert it into a physical or logical standby database in the new configuration. The following sections describe:

- Flashing Back a Failed Primary Database into a Physical Standby Database
- Flashing Back a Failed Primary Database into a Logical Standby Database

---

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

- Flashing Back a Logical Standby Database to a Specific Applied SCN

  See Also: *Oracle Data Guard Broker* for automatic reinstatement of the failed primary database as a new standby database (as an alternative to using Flashback Database)

13.2.1 Flashing Back a Failed Primary Database into a Physical Standby Database

The following steps assume that a failover has been performed to a physical standby database and that Flashback Database was enabled on the old primary database at the time of the failover. This procedure brings the old primary database back into the Data Guard configuration as a 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.

Shut down the old primary database (if necessary), mount it, and flash it back to the value for STANDBY_BECAME_PRIMARY_SCN that was determined in Step 1.

```
SQL> SHUTDOWN IMMEDIATE;
SQL> STARTUP MOUNT;
SQL> FLASHBACK DATABASE TO SCN standby_became_primary_scn;
```

Step 3 Convert the database to a physical standby database.

Perform the following steps on the old primary database:

1. Issue the following statement on the old primary database:

   ```
   SQL> ALTER DATABASE CONVERT TO PHYSICAL STANDBY;
   ```

   This statement will dismount the database after successfully converting the control file to a standby control file.

2. Shut down and restart the database:

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

Step 4 Start transporting redo to the new physical standby database.

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. Issue the following SQL statements on the new primary database:

```
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 redo transport services do not transmit redo data to other databases.

Step 5 Start Redo Apply on the new physical standby database.

Issue the following SQL statement on the new physical standby database:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE
 2> USING CURRENT LOGFILE DISCONNECT;
```

Redo Apply automatically stops each time it encounters a redo record that is generated as the result of a role transition, so Redo Apply will need to be restarted one or more times until it has applied beyond the SCN at which the new primary database became the primary database. 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 8.2.1, "Performing a Switchover to a Physical Standby Database" for more information.

13.2.2 Flashing Back a Failed Primary Database into a Logical Standby Database

These steps assume that the Data Guard configuration has already completed a failover involving a logical standby database and that Flashback Database has been enabled on the old primary database. This procedure brings the old primary database back into the Data Guard configuration as a new logical standby database without having to formally instantiate it from the new primary database.

Step 1 Determine the flashback SCN and the recovery SCN.

The flashback SCN is the SCN to which the failed primary database will be flashed back. The recovery SCN is the SCN to which the failed primary database will be recovered. Issue the following query on the new primary to identify these SCNs:

```
SQL> SELECT merge_change# AS FLASHBACK_SCN, processed_change# AS RECOVERY_SCN
 2> FROM DBA_LOGSTDBY_HISTORY
 3> WHERE stream_sequence# = (SELECT MAX(stream_sequence#)-1
 4> FROM DBA_LOGSTDBY_HISTORY);
```

Step 2 Flash back the failed primary database to the flashback SCN identified in Step 1.

```
SQL> FLASHBACK DATABASE TO SCN flashback_scn;
```

Step 3 Convert the failed primary into a physical standby, and remount the standby database in preparation for recovery.

```
SQL> ALTER DATABASE CONVERT TO PHYSICAL STANDBY;
SQL> SHUTDOWN IMMEDIATE;
SQL> STARTUP MOUNT;
```

Step 4 Identify the logfiles on the new primary that contain redo within range [flashback SCN, recovery SCN].

The logfiles identified by the following query are significant because they are the only "versions" of the archived logfiles that can safely recover the failed primary database. If the logfiles returned from the following query cannot be registered in Step 5, the failed primary can never be revived as a logical standby. In such a case, a logical standby will have to be created from the new primary.

```
SQL> SELECT file_name FROM DBA_LOGSTDBY_LOG
  2> WHERE first_change# <= recovery_scn
  3> AND next_change#  > flashback_scn;
```

Step 5 Register all logfiles returned from Step 4 with the physical standby (failed primary).

```
SQL> ALTER DATABASE REGISTER LOGFILE 'files_from_step _4';
```

Step 6 Recover until the recovery SCN identified in Step 1.

```
SQL> RECOVER MANAGED STANDBY DATABASE UNTIL CHANGE recovery_scn;
```

Step 7 Enable the database guard.

```
SQL> ALTER DATABASE GUARD ALL;
```

Step 8 Activate the physical standby to become a primary database.

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

Step 9 Open the database.

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

Step 10 Create a database link to the new primary, 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.

13.2.3 Flashing Back a Logical Standby Database to a Specific Applied SCN

One of the benefits of a standby database is that Flashback Database can be performed on the standby database without affecting the primary database service. Flashing back a database to a specific point in time is a straightforward task, however on a logical standby database, you may want to flash back to a time just before a known transaction was committed. Such a need can arise when configuring a logical standby database with a new primary database after a failover.

The following steps describe how to use Flashback Database and SQL Apply to recover to a known applied SCN.

Step 1 Once you have determined the known SCN at the primary (APPLIED_SCN), issue the following query to determine the corresponding SCN at the logical standby database, to use for the flashback operation:

```
SQL> SELECT DBMS_LOGSTDBY.MAP_PRIMARY_SCN (PRIMARY_SCN => APPLIED_SCN)
  2> AS TARGET_SCN FROM DUAL;
```

Step 2 Flash back the logical standby to the TARGET_SCN returned.

Issue the following SQL statements to flash back the logical standby database to the specified SCN, and open the logical standby database with the RESETLOGS option:

```
SQL> SHUTDOWN;
SQL> STARTUP MOUNT EXCLUSIVE;
SQL> FLASHBACK DATABASE TO SCN <TARGET_SCN>;
SQL> ALTER DATABASE OPEN RESETLOGS;
```

Step 3 Confirm SQL Apply has applied less than or up to the APPLIED_SCN.

Issue the following query:

```
SQL> SELECT APPLIED_SCN FROM V$LOGSTDBY_PROGRESS;
```

13.3 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 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.)

13.3.1 Flashing Back a Physical Standby Database to a Specific Point-in-Time

The following steps describe how to avoid re-creating a physical standby database after you issued the `OPEN RESETLOGS` statement on the primary database.

Step 1 Determine the SCN before the RESETLOGS operation occurred.

On the primary database, use the following query to obtain the value of the system change number (SCN) that is 2 SCNs before the `RESETLOGS` operation occurred on the primary database:

```
SQL> SELECT TO_CHAR(RESETLOGS_CHANGE# - 2) FROM V$DATABASE;
```

Step 2 Obtain the current SCN on the standby database.

On the standby database, obtain the current SCN with the following query:

```
SQL> SELECT TO_CHAR(CURRENT_SCN) FROM V$DATABASE;
```

Step 3 Determine if it is necessary to flash back the database.

If the value of `CURRENT_SCN` is larger than the value of resetlogs_change# - 2, issue the following statement to flash back the standby database.

```
SQL> FLASHBACK STANDBY DATABASE TO SCN resetlogs_change# -2;
```

- If the value of `CURRENT_SCN` is less than the value of the resetlogs_change# - 2, skip to Step 4.
- If the standby database's SCN is far enough behind the primary database's SCN, apply services will be able to continue through the `OPEN RESETLOGS` statement

without stopping. In this case, flashing back the database is unnecessary because apply services do not stop upon reaching the OPEN RESETLOGS statement in the redo data.

Step 4 Restart Redo Apply.

To start Redo Apply on the physical standby database, issue the following statement:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE
  2> USING CURRENT LOGFILE DISCONNECT;
```

The standby database is now ready to receive and apply redo from the primary database.

13.3.2 Flashing Back a Logical Standby Database to a Specific Point-in-Time

The following steps describe how to avoid re-creating a logical standby database after you have flashed back the primary database and opened it by issuing an OPEN RESETLOGS statement.

> **Note:** If SQL Apply detects the occurrence of a resetlogs operation at the primary database, it automatically mines the correct branch of redo, if it is possible to do so without having to flashback the logical standby database. Otherwise, SQL Apply stops with an error ORA-1346: LogMiner processed redo beyond specified reset log scn. In this section, it is assumed that SQL Apply has already stopped with such an error.

Step 1 Determine the SCN at the primary database.

On the primary database, use the following query to obtain the value of the system change number (SCN) that is 2 SCNs before the RESETLOGS operation occurred on the primary database:

```
SQL> SELECT TO_CHAR(RESETLOGS_CHANGE# - 2) AS FLASHBACK_SCN FROM V$DATABASE;
```

Step 2 Determine the target SCN for flashback operation at the logical standby.

```
SQL> SELECT DBMS_LOGSTDBY.MAP_PRIMARY_SCN (PRIMARY_SCN => FLASHBACK_SCN)
  2> AS TARGET_SCN FROM DUAL;
```

Step 3 Flash back the logical standby to the TARGET_SCN returned.

Issue the following SQL statements to flash back the logical standby database to the specified SCN, and open the logical standby database with the RESETLOGS option:

```
SQL> SHUTDOWN;
SQL> STARTUP MOUNT EXCLUSIVE;
SQL> FLASHBACK DATABASE TO SCN <TARGET_SCN>;
SQL> ALTER DATABASE OPEN RESETLOGS;
```

Step 4 Start SQL Apply.

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

13.4 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.*

13.4.1 Recovery Steps for Logical Standby Databases

For logical standby databases, when SQL Apply encounters a redo record for an operation performed on an interesting table with the `NOLOGGING` clause, it stops with the following error: `ORA-16211 unsupported record found in the archived redo log`.

To recover after the `NOLOGGING` clause is specified, re-create one or more tables from the primary database, as described in Section 10.5.5.

> **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.

13.4.2 Recovery Steps for Physical Standby Databases

When the archived redo log file is copied to the standby site and applied to the physical standby database, a portion of the datafile is unusable and is marked as being unrecoverable. When you either fail over to the physical standby database, or open the standby database for read-only access, and attempt to read the range of blocks that are marked as `UNRECOVERABLE`, you will see error messages similar to the following:

```
ORA-01578: ORACLE data block corrupted (file # 1, block # 2521)
ORA-01110: data file 1: '/oracle/dbs/stdby/tbs_1.dbf'
ORA-26040: Data block was loaded using the NOLOGGING option
```

To recover after the `NOLOGGING` clause is specified, you need to copy the datafile that contains the missing redo data from the primary site to the physical standby site. Perform the following steps:

Step 1 Determine which datafiles should be copied.

Follow these steps:

1. Query the primary database:

```
SQL> SELECT NAME, UNRECOVERABLE_CHANGE# FROM V$DATAFILE;
NAME                                               UNRECOVERABLE
-------------------------------------------------- -------------
/oracle/dbs/tbs_1.dbf                                       5216
/oracle/dbs/tbs_2.dbf                                          0
/oracle/dbs/tbs_3.dbf                                          0
/oracle/dbs/tbs_4.dbf                                          0
4 rows selected.
```

2. Query the standby database:

```
SQL> SELECT NAME, UNRECOVERABLE_CHANGE# FROM V$DATAFILE;
NAME                                               UNRECOVERABLE
-------------------------------------------------- -------------
/oracle/dbs/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 Copy the datafile to the standby database.

Copy the datafile that contains the missing redo data from the primary site to location on the physical standby site where files related to recovery are stored.

Step 4 On the standby database, restart Redo Apply.

Issue the following SQL statement:

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

You might get the following error messages (possibly in the alert log) when you try to restart Redo Apply:

```
ORA-00308: cannot open archived log 'standby1'
ORA-27037: unable to obtain file status
SVR4 Error: 2: No such file or directory
Additional information: 3
ORA-01547: warning: RECOVER succeeded but OPEN RESETLOGS would get error below
ORA-01152: file 1 was not restored from a sufficiently old backup
ORA-01110: data file 1: '/oracle/dbs/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 4. See Section 6.3.3.1 for information about manually resolving an archive gap.

13.4.3 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.

13.5 Creating a Standby Database That Uses OMF or ASM

Chapter 3 and Chapter 4 described how to create physical and logical standby databases. This section augments the discussions in those chapters with additional steps that must be performed if the primary database uses Oracle Managed Files (OMF) or Automatic Storage Management (ASM).

Note: The discussion in this section is presented at a level of detail that assumes the reader already knows how to create a physical standby database and is an experienced user of the RMAN, OMF, and ASM features. For more information, see:

- Chapter 3, Chapter 4, and Appendix F for information about creating physical and logical standby databases
- *Oracle Database Administrator's Guide* for information about OMF and ASM
- *Oracle Database Backup and Recovery User's Guide* and *Oracle Database Backup and Recovery Reference* for information about RMAN

Perform the following tasks to prepare for standby database creation:

1. Enable forced logging on the primary database.
2. Enable archiving on the primary database.
3. Set all necessary initialization parameters on the primary database.
4. Create an initialization parameter file for the standby database.
5. If the primary database is configured to use OMF, then Oracle recommends that the standby database be configured to use OMF, too. To do this, set the DB_CREATE_FILE_DEST and DB_CREATE_ONLINE_LOG_DEST_*n* initialization

parameters to appropriate values. Maintenance and future role transitions are simplified if the same disk group names are used for both the primary and standby databases.

6. Set the STANDBY_FILE_MANAGEMENT initialization parameter to AUTO.
7. Configure Oracle Net, as required, to allow connections to the standby database.
8. Configure redo transport authentication as described in Section 3.1.2, "Configure Redo Transport Authentication".
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@lstdby

target database Password: password

RMAN> CATALOG START WITH '+stby_diskgroup';
RMAN> SWITCH DATABASE TO COPY;
```

After you successfully complete these steps, continue with the steps in Section 4.2.5 to start, recover, and verify the logical standby database.

13.6 Recovering From Lost-Write Errors on a Primary Database

During media recovery in a Data Guard configuration, a physical standby database can be used to detect lost-write data corruption errors on the primary database. This is done by comparing SCNs of blocks stored in the redo log on the primary database to SCNs of blocks on the physical standby database. If the SCN of the block on the primary database is lower than the SCN on the standby database, then there was a lost-write error on the primary database.

> **Note:** Because lost-write errors are detected only when a block is read into the cache by a primary and the corresponding redo is later compared to the block on the standby, there may be undetected stale blocks on both the primary and the standby that have not yet been read and verified. These stale blocks do not affect operation of the current database because until those blocks are read, all blocks that have been used up to the SCN of the currently applied redo on the standby to do queries or updates were verified by the standby.

When a primary lost-write error is detected on the standby, one or more block error messages similar to the following for each stale block are printed in the alert file of the standby database:

```
Tue Dec 12 19:09:48 2006
STANDBY REDO APPLICATION HAS DETECTED THAT THE PRIMARY DATABASE
LOST A DISK WRITE OF BLOCK 26, FILE 7
NO REDO AT OR AFTER SCN 389667 CAN BE USED FOR RECOVERY.
.
.
.
```

The alert file then shows that an `ORA-00752` error is raised on the standby database and the managed recovery is cancelled:

```
Slave exiting with ORA-752 exception
Errors in file /oracle/log/diag/rdbms/dgstwrite2/stwrite2/trace/stwrite2_pr00_
23532.trc:
ORA-00752: recovery detected a lost write of a data block
ORA-10567: Redo is inconsistent with data block (file# 7, block# 26)
ORA-10564: tablespace TBS_2
ORA-01110: data file 7: '/oracle/dbs/btbs_21.f'
ORA-10561: block type 'TRANSACTION MANAGED DATA BLOCK', data object# 57503
.
.
.
```

The standby database is then recovered to a consistent state, without any corruption to its datafiles caused by this error, at the SCN printed in the alert file:

```
Recovery interrupted!
Recovered data files to a consistent state at change 389569
```

This last message may appear significantly later in the alert file and it may have a lower SCN than the block error messages. Also, the primary database may operate without visible errors even though its datafiles may already be corrupted.

The recommended procedure to recover from such errors is a failover to the physical standby, as described in the following steps.

Steps to Failover to a Physical Standby After Lost-Writes Are Detected on the Primary

1. Shut down the primary database. All data at or after SCN printed in the block error messages will be lost.
2. Issue the following SQL statement on the standby database to convert it to a primary:

```
SQL> ALTER DATABASE ACTIVATE STANDBY DATABASE;

Database altered.

Tue Dec 12 19:15:23 2006
alter database activate standby database
ALTER DATABASE ACTIVATE [PHYSICAL] STANDBY DATABASE (stwrite2)
RESETLOGS after incomplete recovery UNTIL CHANGE 389569
Resetting resetlogs activation ID 612657558 (0x24846996)
Online log /oracle/dbs/bt_log1.f: Thread 1 Group 1 was previously cleared
Online log /oracle/dbs/bt_log2.f: Thread 1 Group 2 was previously cleared
Standby became primary SCN: 389567
Tue Dec 12 19:15:23 2006
Setting recovery target incarnation to 3
Converting standby mount to primary mount.
ACTIVATE STANDBY: Complete - Database mounted as primary (stwrite2)
Completed: alter database activate standby database
```

3. Back up the new primary. Performing a backup immediately is a necessary safety measure, because you cannot recover changes made after the failover without a complete backup copy of the database. As a result of the failover, the original primary database can no longer participate in the Data Guard configuration, and all other standby databases will now receive and apply redo data from the new primary database.
4. Open the new primary database.
5. An optional step is to recreate the failed primary as a physical standby. This can be done using the database backup taken at the new primary in step 3. (You cannot use flashback database or the Data Guard broker to reinstantiate the old primary database in this situation.)

   Be aware that a physical standby created using the backup taken from the new primary will have the same datafiles as the old standby. Therefore, any undetected lost writes that the old standby had before it was activated will not be detected by the new standby, since the new standby will be comparing the same blocks. Any new lost writes that happen on either the primary or the standby will be detected.

   See Also: *Oracle Database Backup and Recovery User's Guide* for more information about enabling lost-write detection

13.7 Converting a Failed Primary into a Standby Database Using RMAN Backups

To convert a failed primary database, Oracle recommends that you enable the Flashback Database feature on the primary and follow the procedure described in either Section 13.2.1 or Section 13.2.2. The procedures in those sections describe the fastest ways to convert a failed primary into either a physical or logical standby. However, if Flashback Database was not enabled on the failed primary, you can still

convert the failed primary into either a physical or logical standby using a local backup of the failed primary, as described in the following sections:

- Converting a Failed Primary into a Physical Standby Using RMAN Backups
- Converting a Failed Primary into a Logical Standby Using RMAN Backups

13.7.1 Converting a Failed Primary into a Physical Standby Using RMAN Backups

The steps in this section describe how to convert a failed primary into a physical standby by using RMAN backups. This procedure requires that the `COMPATIBLE` initialization parameter of the old primary be set to at least 11.0.0.

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 Restore and recover the entire database.

Restore the database with a backup taken before the old primary had reached the SCN at which the standby became the new primary (`standby_became_primary_scn`). Then, perform a point-in-time recovery to recover the old primary to that same point.

Issue the following RMAN commands:

```
RMAN> RUN
    {
      SET UNTIL SCN <standby_became_primary_scn + 1>;
      RESTORE DATABASE;
      RECOVER DATABASE;
     }
```

With user-managed recovery, you can first restore the database manually. Typically, a backup taken a couple of hours before the failover would be old enough. You can then recover the failed primary using the following command:

```
SQL> RECOVER DATABASE USIING BACKUP CONTROLFILE UNTIL CHANGE
<standby_became_primary_scn + 1>;
```

Unlike a reinstantiation that uses Flashback Database, this procedure adds one to `standby_became_primary_scn`. For datafiles, flashing back to an SCN is equivalent to recovering up until that SCN plus one.

Step 3 Convert the database to a physical standby database.

Perform the following steps on the old primary database:

1. Issue the following statement on the old primary database:

   ```
   SQL> ALTER DATABASE CONVERT TO PHYSICAL STANDBY;
   ```

   This statement will dismount the database after successfully converting the control file to a standby control file.

2. Shut down and restart the database:

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

Step 4 Open the database as read-only.

Issue the following command:

```
SQL> ALTER DATABASE OPEN READ ONLY;
```

The goal of this step is to synchronize the control file with the database by using a dictionary check. After this command, check the alert log for any actions suggested by the dictionary check. Typically, no user action is needed if the old primary was not in the middle of adding or dropping datafiles during the failover.

Step 5 (Optional) Mount the standby again, if desired

A physical standby can apply redo while it is open read-only. But if you plan to recover the physical standby without opening it read-only, you may optionally shut it down and mount it again, as follows:

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

Step 6 Restart transporting redo to the new physical standby database.

Before the new standby database was created, the new primary database probably stopped transmitting redo to the remote destination. To restart redo transport services, perform the following steps on the new primary database:

1. Issue the following query to see the current state of the archive destinations:

   ```
   SQL> SELECT DEST_ID, DEST_NAME, STATUS, PROTECTION_MODE, DESTINATION, ERROR,SRL
     2> FROM V$ARCHIVE_DEST_STATUS;
   ```

2. If necessary, enable the destination:

   ```
   SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_n=ENABLE;
   ```

3. Perform a log switch to ensure the standby database begins receiving redo data from the new primary database, and verify it was sent successfully.

   > **Note:** This is an important step in order for the old primary to become a new standby following the new primary. If this step is not done, the old primary may recover to an incorrect database branch. The only way to correct the problem then is to convert the old primary again.

   At the SQL prompt, enter the following statements:

   ```
   SQL> ALTER SYSTEM SWITCH LOGFILE;
   SQL> SELECT DEST_ID, DEST_NAME, STATUS, PROTECTION_MODE, DESTINATION, ERROR,SRL
     2> FROM V$ARCHIVE_DEST_STATUS;
   ```

   On the new standby database, you may also need to change the LOG_ARCHIVE_DEST_*n* initialization parameters so that redo transport services do not transmit redo data to other databases. This step can be skipped if both the primary and standby database roles were set up with the VALID_FOR attribute in one server parameter file (SPFILE). By doing this, the Data Guard configuration operates properly after a role transition.

Step 7 Start Redo Apply.

Start Redo Apply on the new physical standby database, as follows:

```
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 8.2.1, "Performing a Switchover to a Physical Standby Database" for more information.

13.7.2 Converting a Failed Primary into a Logical Standby Using RMAN Backups

The steps in this section describe how to convert a failed primary into a logical standby using RMAN backups.

Step 1 Determine the SCN to which to recover the failed primary database.

On the new primary database, issue the following query to determine the SCN to which you want to recover the failed primary database:

```
SQL> SELECT APPLIED_SCN RECOVERY_SCN FROM V$LOGSTDBY_PROGRESS;
```

Also on the new primary database, determine the SCN to use in dealing with archive logs, as follows:

1. Ensure all standby redo logs have been archived. Issue the following query, looking for a value of `READY` to be returned. Depending on the size of the database and the number of logs needing to be archived, it could take some time before a status of `READY` is returned.

   ```
   SQL> SELECT VALUE FROM SYSTEM.LOGSTDBY$PARAMETERS WHERE NAME='REINSTATEMENT_
   STATUS';
   ```

2. After a status of `READY` has been returned, run the following query to retrieve the SCN for dealing with archive logs as part of this recovery:

   ```
   SQL> SELECT VALUE ARCHIVE_SCN FROM SYSTEM.LOGSTDBY$PARAMETERS
     2> WHERE NAME='STANDBY_BECAME_PRIMARY_SCN';
   ```

Step 2 Remove divergent archive logs from the failed primary database.

Remove any archive logs created at the time of, or after the failover operation, from the failed primary database. If the failed primary database was isolated from the standby, it could have divergent archive logs that are not consistent with the current primary database. To ensure these divergent archive logs are never applied, they must be deleted from backups and the flash recovery area. You can use the following RMAN command to delete the relevant archive logs from the flash recovery area:

```
RMAN> DELETE ARCHIVELOG FROM SCN ARCHIVE_SCN;
```

Once deleted, these divergent logs and subsequent transactions can never be recovered.

Step 3 Determine the log files to be copied to the failed primary database.

On the new primary database, issue the following query to determine the minimum set of log files that must be copied to the failed primary database before recovering from a backup:

```
SQL> SELECT file_name FROM DBA_LOGSTDBY_LOG WHERE next_change# > ARCHIVE_SCN;
```

Retrieve the required standby logs, copy the backup set to the new standby and restore it to the new standby flash recovery area. Because these logs are coming from

standby redo logs, they are not part of the standby's standard archives. The RMAN utility is able to use a partial file name to retrieve the files from the correct location.

The following is a sample use of the RMAN BACKUP command:

```
RMAN> BACKUP AS COPY DEVICE TYPE DISK FORMAT '/tmp/test/%U'
> ARCHIVELOG LIKE '<partial file names from above>%';
```

The following is a sample use of the RMAN RESTORE command:

```
RMAN> CATALOG START WITH '/tmp/test';
RMAN> RESTORE ARCHIVELOG FROM SEQUENCE 33 UNTIL SEQUENCE 35;
```

Step 4 Restore a backup and recover the database.

Restore a backup of all the original primary's data files and recover to RECOVERY_SCN + 1. Oracle recommends that you leverage the current control file.

1. Start up the database in restricted mode to protect it from rogue transactions until the GUARD ALL command can be issued after the database has been opened.
2. Use the backup to restore the data files of the failed primary database.
3. Turn off flashback database, if it is enabled (necessary for the USING BACKUP CONTROLFILE clause).
4. Perform point-in-time recovery to RECOVERY_SCN +1 in SQL*Plus.

Whether you are using a current control file or a backup control file, you must specify the USING BACKUP CONTROLFILE clause to allow you to point to the archive logs being restored. Otherwise, the recovery process could attempt to access online redo logs instead of the logs retrieved in Step 3. When prompted for the sequences retrieved in Step 3, ensure you specify the file names of the restored archive log copies, as follows:

```
SQL> RECOVER DATABASE UNTIL CHANGE RECOVERY_SCN + 1 USING BACKUP CONTROLFILE;
```

Step 5 Open the database with the RESETLOGS option.

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

Step 6 Enable Database Guard

```
SQL> ALTER DATABASE GUARD ALL;
```

Step 7 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;
```

At this point, you can disable restricted session (ALTER SYSTEM DISABLE RESTRICTED SESSION) or, if you need to restart the database to re-enable Flashback from Step 4.3, let this restart turn off RESTRICTED SESSION.

Part II

Reference

This part provides reference material to be used in conjunction with the Oracle Data Guard standby database features.

This part contains the following chapters:

14

Initialization Parameters

This chapter describes the initialization parameters that affect databases in a Data Guard environment.

Table 14–1 lists the initialization parameters and indicates if the parameter applies to the primary database role, the standby database role, or both. The table also includes notes and recommendations specific to setting the parameters in a Data Guard environment. *Oracle Database Reference* provides complete initialization parameter information, including how to update initialization parameters by issuing the `ALTER SYSTEM SET` statement (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 14–1 Initialization Parameters for Instances in a Data Guard Configuration

Parameter	Applicable To	Notes and Recommendations
`COMPATIBLE` = *release_number*.	Primary Logical Standby Physical Standby Snapshot Standby	Specify the same value on the primary and standby databases if you expect to do a switchover. If the values differ, redo transport services may be unable to transmit redo data from the primary database to the standby databases. See Section 3.2.3 for an example. For rolling upgrades using SQL Apply, set this parameter according to the guidelines described in Section 12.4, "Performing a Rolling Upgrade By Creating a New Logical Standby Database".
`CONTROL_FILE_RECORD_KEEP_TIME` = *number_of_days*	Primary Logical Standby Physical Standby Snapshot Standby	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).
`CONTROL_FILES` = '*control_file_name*' , *control_file_name*', '...')	Primary Logical Standby Physical Standby Snapshot Standby	Required. Specify the path name and filename for one or more control files. The control files must already exist on the database. Oracle recommends using 2 control files. If another copy of the current control file is available, then an instance can be easily restarted after copying the good control file to the location of the bad control file. See Section 3.2.3 for an example.

Table 14–1 (Cont.) Initialization Parameters for Instances in a Data Guard Configuration

Parameter	Applicable To	Notes and Recommendations
DB_FILE_NAME_CONVERT = (*location_of_primary_database_datafile*' , '*location_of_standby_database_datafile_name*' , '...'	Physical Standby Snapshot Standby	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 3–1.
DB_UNIQUE_NAME = *Unique name for the database*	Primary Logical Standby Physical Standby Snapshot Standby	Recommended, but required if you specify the LOG_ARCHIVE_CONFIG parameter. Specifies a unique name for this database. This name does not change even if the primary and standby databases reverse roles. The DB_UNIQUE_NAME parameter defaults to the value of the DB_NAME parameter.
FAL_CLIENT = *Oracle_Net_service_name*	Physical Standby Snapshot Standby	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).
FAL_SERVER = *Oracle_Net_service_name*	Physical Standby Snapshot Standby	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.
INSTANCE_NAME	Primary Logical Standby Physical Standby Snapshot Standby	Optional. If this parameter is defined and the primary and standby databases reside on the same host, specify a different name for the standby database than you specify for the primary database. See Section 3.2.3 for an example.
LOG_ARCHIVE_CONFIG='DG_CONFIG=(*db_unique_name*, *db_unique_name*, ...)'	Primary Logical Standby Physical Standby Snapshot Standby	Recommended. Specify the DG_CONFIG attribute to identify the DB_UNIQUE_NAME for the primary database and each standby database in the Data Guard configuration. The DG_CONFIG attribute must be set to enable the dynamic addition of a standby database to a Data Guard configuration that has an Oracle Real Application Clusters (RAC) primary database running in either maximum protection or maximum availability mode.
LOG_ARCHIVE_DEST_*n* = {LOCATION=*path_name* | SERVICE=*service_name*, *attribute*, *attribute*, ... }	Primary Logical Standby Physical Standby Snapshot Standby	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.
LOG_ARCHIVE_DEST_STATE_*n* = {ENABLE | DEFER | ALTERNATE}	Primary Logical Standby Physical Standby Snapshot Standby	Required. Specify a LOG_ARCHIVE_DEST_STATE_*n* parameter to enable or disable redo transport services to transmit redo data to the specified (or to an alternate) destination. Define a LOG_ARCHIVE_DEST_STATE_*n* parameter for every LOG_ARCHIVE_DEST_*n* parameter. See also Chapter 15.
LOG_ARCHIVE_FORMAT=*log%d_%t_%s_%r*.arc	Primary Logical Standby Physical Standby Snapshot Standby	The LOG_ARCHIVE_FORMAT and LOG_ARCHIVE_DEST_n parameters are concatenated together to generate fully qualified archived redo log filenames on a database.

Table 14–1 (Cont.) Initialization Parameters for Instances in a Data Guard Configuration

Parameter	Applicable To	Notes and Recommendations
LOG_ARCHIVE_LOCAL_FIRST =[TRUE|FALSE]	Primary Snapshot Standby	This parameter has been deprecated and is maintained for backward compatibility only. Oracle recommends that this parameter only be set to TRUE if it is explicitly set.
LOG_ARCHIVE_MAX_PROCESSES =*integer*	Primary Logical Standby Physical Standby Snapshot Standby	Optional. Specify the number (from 1 to 30) of archiver (ARC*n*) processes you want Oracle software to invoke initially. The default value is 4.
LOG_ARCHIVE_TRACE=*integer*	Primary Logical Standby Physical Standby Snapshot Standby	Optional. Set this parameter to trace the transmission of redo data to the standby site. The valid integer values are described in Appendix G.
LOG_FILE_NAME_CONVERT ='*location_of_primary_database_redo_logs*','*location_of_standby_database_redo_logs*'	Logical Standby Physical Standby Snapshot Standby	Required when the standby database is on the same system as the primary database or when the directory structure where the log files are located on the standby site is different from the primary site. This parameter converts the path names of the primary database online redo log file to path names on the standby database. See Section 3.2.3 for an example.
REMOTE_LOGIN_ PASSWORDFILE= {EXCLUSIVE|SHARED]	Primary Logical Standby Physical Standby Snapshot Standby	Optional if operating system authentication is used for administrative users and SSL is used for redo transport authentication. Otherwise, this parameter must be set to EXCLUSIVE or SHARED on every database in a Data Guard configuration.
SHARED_POOL_SIZE = *bytes*	Primary Logical Standby Physical Standby Snapshot Standby	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.
STANDBY_ARCHIVE_DEST= *filespec*	Logical Standby Physical Standby Snapshot Standby	This parameter has been deprecated and is maintained for backward compatibility only.
STANDBY_FILE_MANAGEMENT = {AUTO|MANUAL}	Primary Physical Standby Snapshot Standby	Set the STANDBY_FILE_MANAGEMENT parameter to AUTO so that when datafiles are added to or dropped from the primary database, corresponding changes are made in the standby database without manual intervention. If the directory structures on the primary and standby databases are different, you must also set the DB_FILE_NAME_CONVERT initialization parameter to convert the filenames of one or more sets of datafiles on the primary database to filenames on the (physical) standby database. See Example 3–1 for more information and examples.

15

LOG_ARCHIVE_DEST_n Parameter Attributes

This chapter provides reference information for the attributes of the LOG_ARCHIVE_DEST_*n* initialization parameter. The following list shows the attributes:

Each LOG_ARCHIVE_DEST_*n* destination must contain either a LOCATION or SERVICE attribute to specify a local disk directory or a remotely accessed database, respectively. All other attributes are optional.

Note: Several attributes of the LOG_ARCHIVE_DEST_*n* initialization parameter have been deprecated. These attributes are supported for backward compatibility only and are documented in the *Oracle Database Reference*.

See Also: Chapter 6 for more information about defining LOG_ARCHIVE_DEST_*n* destinations and setting up redo transport services

AFFIRM and NOAFFIRM

Controls whether a redo transport destination acknowledges received redo data before or after writing it to the standby redo log:

- AFFIRM—specifies that a redo transport destination acknowledges received redo data *after* writing it to the standby redo log.
- NOAFFIRM—specifies that a redo transport destination acknowledges received redo data *before* writing it to the standby redo log.

Category	AFFIRM	NOAFFIRM
Data type	Keyword	Keyword
Valid values	Not applicable	Not applicable
Default Value	Not applicable	Not applicable
Requires attributes	SERVICE	SERVICE
Conflicts with attributes	NOAFFIRM	AFFIRM
Corresponds to	AFFIRM column of the V$ARCHIVE_DEST view	AFFIRM column of the V$ARCHIVE_DEST view

Usage Notes

- If neither the AFFIRM nor the NOAFFIRM attribute is specified, the default is AFFIRM when the SYNC attribute is specified and NOAFFIRM when the ASYNC attribute is specified.
- Specification of the AFFIRM attribute without the SYNC attribute is deprecated and will not be supported in future releases.

See also: SYNC and ASYNC attributes on page 15-20

Examples

The following example shows the AFFIRM attribute for a remote destination.

```
LOG_ARCHIVE_DEST_3='SERVICE=stby1 SYNC AFFIRM'
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

ALTERNATE

Specifies an alternate archiving destination to be used when the original destination fails.

Category	ALTERNATE=LOG_ARCHIVE_DEST_*n*
Data Type	String
Valid Value	A LOG_ARCHIVE_DEST_*n* destination
Default Value	None. If an alternate destination is not specified, then redo transport services do not automatically change to another destination.
Requires attributes	Not applicable
Conflicts with attributes	None [1]
Corresponds to	ALTERNATE and STATUS columns of the V$ARCHIVE_DEST view

[1] If the REOPEN attribute is specified with a nonzero value, the ALTERNATE attribute is ignored. If the MAX_FAILURE attribute is also specified with a nonzero value, and the failure count exceeds the specified failure threshold, the ALTERNATE destination is enabled. Therefore, the ALTERNATE attribute does not conflict with a nonzero REOPEN attribute value.

Usage Notes

- The ALTERNATE attribute is optional. If an alternate destination is not specified, then redo transport services do not automatically change to another destination if the original destination fails.
- You can specify only one alternate destination for each LOG_ARCHIVE_DEST_*n* parameter, but several enabled destinations can share the same alternate destination.
- Ideally, an alternate destination should specify either:
  - A different disk location on the same local standby database system (shown in Example 15–1 on page 15-4)
  - A different network route to the same standby database system (shown in Example 15–2 on page 15-4)
  - A remote standby database system that closely mirrors that of the enabled destination
- If no enabled destinations reference an alternate destination, the alternate destination is implied to be deferred, because there is no automatic method of enabling the alternate destination. However, you can enable (or defer) alternate destinations at runtime using either ALTER SYSTEM.
- Any destination can be designated as an alternate destination, given the following restrictions:
  - At least one local mandatory destination is enabled.
  - The number of enabled destinations must meet the defined LOG_ARCHIVE_MIN_SUCCEED_DEST parameter value.
  - A destination cannot be its own alternate.

- Increasing the number of enabled destinations decreases the number of available alternate archiving destinations.
- When a destination fails, its alternate destination is enabled on the next archival operation. There is no support for enabling the alternate destination in the middle of the archival operation because that would require rereading already processed blocks. This is identical to the REOPEN attribute behavior.
- If the REOPEN attribute is specified with a nonzero value, the ALTERNATE attribute is ignored unless the MAX_FAILURE attribute has a nonzero value. If the MAX_FAILURE and REOPEN attributes have nonzero values and the failure count exceeds the specified failure threshold, the ALTERNATE destination is enabled. Therefore, the ALTERNATE attribute does not conflict with a nonzero REOPEN attribute value.

Examples

In the sample initialization parameter file in Example 15–1, 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 15–1 Automatically Failing Over to an Alternate Destination

```
LOG_ARCHIVE_DEST_1='LOCATION=/disk1 MANDATORY ALTERNATE=LOG_ARCHIVE_DEST_2'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_2='LOCATION=/disk2 MANDATORY'
LOG_ARCHIVE_DEST_STATE_2=ALTERNATE
```

Notice in the example that a destination can also be in the ALTERNATE *state*, as specified with the LOG_ARCHIVE_DEST_STATE_*n* initialization parameter. The ALTERNATE state defers redo transport services from transmitting redo data to this destination until such time as another destination failure automatically enables this destination.

Example 15–2 Defining an Alternate Oracle Net Service Name to the Same Standby Database

This example shows how to define an alternate Oracle Net service name to the same standby database.

```
LOG_ARCHIVE_DEST_1='LOCATION=/disk1 MANDATORY'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_2='SERVICE=stby1_path1 ALTERNATE=LOG_ARCHIVE_DEST_3'
LOG_ARCHIVE_DEST_STATE_2=ENABLE
LOG_ARCHIVE_DEST_3='SERVICE=stby1_path2'
LOG_ARCHIVE_DEST_STATE_3=ALTERNATE
```

COMPRESSION

The COMPRESSION attribute is used to specify whether redo data is transmitted to a redo transport destination in compressed form or uncompressed form when resolving redo data gaps.

> **Note:** Redo transport compression is a feature of the Oracle Advanced Compression option. You must purchase a license for this option before using the redo transport compression feature.

Category	COMPRESSION=ENABLE or DISABLE
Data Type	Boolean
Valid values	ENABLE or DISABLE
Default value	DISABLE
Requires attributes	None
Conflicts with attributes	None
Corresponds to	COMPRESSION column of the V$ARCHIVE_DEST view

Usage Notes

- The COMPRESSION attribute is optional. If it is not specified, the default compression behavior is DISABLE.

Example

The following example shows the COMPRESSION attribute with the LOG_ARCHIVE_DEST_*n* parameter.

```
LOG_ARCHIVE_DEST_3='SERVICE=denver SYNC COMPRESSION=ENABLE'
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

DB_UNIQUE_NAME

Specifies a unique name for the database at this destination.

Category	DB_UNIQUE_NAME=*name*
Data Type	String
Valid values	The name must match the value that was defined for this database with the DB_UNIQUE_NAME parameter.
Default value	None
Requires attributes	None
Conflicts with attributes	None
Corresponds to	DB_UNIQUE_NAME column of the V$ARCHIVE_DEST view

Usage Notes

- This attribute *is optional* if:
  - The LOG_ARCHIVE_CONFIG=DG_CONFIG initialization parameter is not specified.
  - This is a local destination (specified with the LOCATION attribute).
- This attributes *is required* if the LOG_ARCHIVE_CONFIG=DG_CONFIG initialization parameter is specified and if this is a remote destination (specified with the SERVICE attribute).
- Use the DB_UNIQUE_NAME attribute to clearly identify the relationship between a primary and standby databases. This attribute is particularly helpful if there are multiple standby databases in the Data Guard configuration.
- The name specified by the DB_UNIQUE_NAME must match one of the DB_UNIQUE_NAME values in the DG_CONFIG list. Redo transport services validate that the DB_UNIQUE_NAME attribute of the database at the specified destination matches the DB_UNIQUE_NAME attribute or the connection to that destination is refused.
- The name specified by the DB_UNIQUE_NAME attribute must match the name specified by the DB_UNIQUE_NAME initialization parameter of the database identified by the destination.

Example

In the following example, the DB_UNIQUE_NAME parameter specifies boston (DB_UNIQUE_NAME=boston), which is also specified with the DB_UNIQUE_NAME attribute on the LOG_ARCHIVE_DEST_1 parameter. The DB_UNIQUE_NAME attribute on the LOG_ARCHIVE_DEST_2 parameter specifies the chicago destination. Both boston and chicago are listed in the LOG_ARCHIVE_CONFIG=DG_CONFIG parameter.

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

DELAY

Specifies a time lag between when redo data is archived on a standby site and when the archived redo log file is applied to the standby database.

Category	DELAY[=*minutes*]
Data Type	Numeric
Valid values	>=0 minutes
Default Value	30 minutes
Requires attributes	SERVICE
Conflicts with attributes	LOCATION
Corresponds to	DELAY_MINS and DESTINATION columns of the V$ARCHIVE_DEST view

Usage Notes

- The DELAY attribute is optional. By default there is no delay.
- The DELAY attribute indicates the archived redo log files at the standby destination are not available for recovery until the specified time interval has expired. The time interval is expressed in minutes, and it starts when the redo data is successfully transmitted to, and archived at, the standby site.
- The DELAY attribute may be used to protect a standby database from corrupted or erroneous primary data. However, there is a tradeoff because during failover it takes more time to apply all of the redo up to the point of corruption.
- The DELAY attribute does not affect the transmittal of redo data to a standby destination.
- If you have real-time apply enabled, any delay that you set will be ignored.
- Changes to the DELAY attribute take effect the next time redo data is archived (after a log switch). In-progress archiving is not affected.
- You can override the specified delay interval at the standby site, as follows:
  - For a physical standby database:

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

  - For a logical standby database:

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

    See Also: *Oracle Database SQL Language Reference* for more information about these ALTER DATABASE statements

Examples

You can use the DELAY attribute to set up a configuration where multiple standby databases are maintained in varying degrees of synchronization with the primary database. However, this protection incurs some overhead during failover, because it takes Redo Apply more time to apply all the redo up to the corruption point.

For example, assume primary database A has standby databases B and C. Standby database B is set up as the disaster recovery database and therefore has no time lag.

Standby database C is set up with a 2-hour delay, which is enough time to allow user errors to be discovered before they are propagated to the standby database.

The following example shows how to specify the DELAY attribute for this configuration:

```
LOG_ARCHIVE_DEST_1='LOCATION=/oracle/dbs/'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_2='SERVICE=stbyB SYNC AFFIRM'
LOG_ARCHIVE_DEST_STATE_2=ENABLE
LOG_ARCHIVE_DEST_3='SERVICE=stbyC DELAY=120'
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

Note: Alternatively, you can use Flashback Database to revert the database to a point-in-time or SCN in a different database incarnation as long as there is sufficient flashback log data. Using Flashback Database is described in *Oracle Database Backup and Recovery User's Guide*.

LOCATION and SERVICE

Each destination *must* specify either the LOCATION or the SERVICE attribute to identify either a local disk directory or a remote database destination where redo transport services can transmit redo data.

Category	LOCATION=*local_disk_directory* or USE_DB_RECOVERY_FILE_DEST	SERVICE=*net_service_name*
Data type	String value	String value
Valid values	Not applicable	Not applicable
Default Value	None	None
Requires attributes	Not applicable	Not applicable
Conflicts with attributes	SERVICE, DELAY, NOREGISTER, SYNC, ASYNC, NET_TIMEOUT, AFFIRM, NOAFFIRM, COMPRESSION, MAX_CONNECTIONS	LOCATION
Corresponds to	DESTINATION and TARGET columns of the V$ARCHIVE_DEST view	DESTINATION and TARGET columns of the V$ARCHIVE_DEST view

Usage Notes

- Either the LOCATION or the SERVICE attribute must be specified. There is no default.
- If you are specifying multiple attributes, specify the LOCATION or SERVICE attribute first in the list of attributes.
- You must specify at least one local disk directory with the LOCATION attribute. This ensures that local archived redo log files are accessible should media recovery of a database be necessary. You can specify up to nine additional local or remote destinations.
- For the LOCATION attribute, you can specify one of the following:
  - LOCATION=*local_disk_directory*

    This specifies a unique directory path name for a disk directory on the system that hosts the database. This is the local destination for archived redo log files.
  - LOCATION=USE_DB_RECOVERY_FILE_DEST

    To configure a flash recovery area, specify the directory or Oracle Storage Manager disk group that will serve as the flash recovery area using the DB_RECOVERY_FILE_DEST initialization parameter. For more information about flash recovery areas, see *Oracle Database Backup and Recovery User's Guide*.
- When you specify a SERVICE attribute:
  - You identify remote destinations by specifying the SERVICE attribute with a valid Oracle Net service name (SERVICE=*net_service_name*) that identifies the remote Oracle database instance to which the redo data will be sent.

    The Oracle Net service name that you specify with the SERVICE attribute is translated into a connection descriptor that contains the information necessary for connecting to the remote database.

See Also: *Oracle Database Net Services Administrator's Guide* for details about setting up Oracle Net service names

  - Transmitting redo data to a remote destination requires a network connection and an Oracle database instance associated with the remote destination to receive the incoming redo data.

- To verify the current settings for `LOCATION` and `SERVICE` attributes, query the `V$ARCHIVE_DEST` fixed view:
  - The `TARGET` column identifies if the destination is local or remote to the primary database.
  - The `DESTINATION` column identifies the values that were specified for a destination. For example, the destination parameter value specifies the Oracle Net service name identifying the remote Oracle instance where the archived redo log files are located.

Examples

Example 1 Specifying the LOCATION Attribute

```
LOG_ARCHIVE_DEST_2='LOCATION=/disk1/oracle/oradata/payroll/arch/'
LOG_ARCHIVE_DEST_STATE_2=ENABLE
```

Example 2 Specifying the SERVICE Attribute

```
LOG_ARCHIVE_DEST_3='SERVICE=stby1'
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

MANDATORY

Specifies that filled online log files must be successfully archived to the destination before they can be reused.

Category	MANDATORY
Data type	Keyword
Valid values	Not applicable
Default value	Not applicable
Requires attributes	Not applicable
Conflicts with attributes	Optional
Corresponds to	BINDING column of the V$ARCHIVE_DEST view

Usage Notes

- If MANDATORY is not specified, then, by default, the destination is considered to be optional.

  At least one destination must succeed, even if all destinations are optional. If archiving to an optional destination fails, the online redo log file is still available for reuse and may be overwritten eventually. However, if the archival operation of a mandatory destination fails, online redo log files cannot be overwritten.

- The LOG_ARCHIVE_MIN_SUCCEED_DEST=*n* parameter (where *n* is an integer from 1 to 10) specifies the number of destinations that must archive successfully before online redo log files can be overwritten.

  All MANDATORY destinations and optional local destinations contribute to satisfying the LOG_ARCHIVE_MIN_SUCCEED_DEST=*n* count. If the value set for the LOG_ARCHIVE_MIN_SUCCEED_DEST parameter is met, the online redo log file is available for reuse. For example, you can set the parameter as follows:

  ```
  # Database must archive to at least two locations before
  # overwriting the online redo log files.
  LOG_ARCHIVE_MIN_SUCCEED_DEST = 2
  ```

- You must have at least one local destination, which you can declare MANDATORY or leave as optional.

  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 makes the LOG_ARCHIVE_MIN_SUCCEED_DEST parameter irrelevant.

- The LOG_ARCHIVE_MIN_SUCCEED_DEST parameter value cannot be greater than the number of mandatory destinations plus the number of optional local destinations.

- The BINDING column of the V$ARCHIVE_DEST fixed view specifies how failure affects the archival operation

Examples

The following example shows the MANDATORY attribute:

```
LOG_ARCHIVE_DEST_1='LOCATION=/arch/dest MANDATORY'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_3='SERVICE=denver MANDATORY'
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

MAX_CONNECTIONS

Enables multiple network connections to be used when sending an archived redo log file to a redo transport destination. Using multiple network connections can improve redo transport performance over high-latency network links.

Category	Description
Data type	Integer
Valid values	1 to 5
Default value	1
Requires attributes	None
Conflicts with attributes	None
Corresponds to	MAX_CONNECTIONS column of the V$ARCHIVE_DEST view of the primary database

Usage Notes

- The MAX_CONNECTIONS attribute is optional. If it is specified, it is only used when redo transport services use ARC*n* processes for archival.
  - If MAX_CONNECTIONS is set to 1 (the default), redo transport services use a single ARC*n* process to transmit redo data to the remote destination.
  - If MAX_CONNECTIONS is set to a value greater than 1, redo transport services use multiple ARC*n* processes to transmit redo in parallel to archived redo log files at the remote destination. Each archiver (ARC*n*) process uses a separate network connection.
- With multiple ARC*n* processes, redo transmission occurs in parallel, thus increasing the speed at which redo is transmitted to the remote destination.
- Any standby database using archiver (ARC*n*) processes will not use standby redo logs if the MAX_CONNECTIONS attribute is specified. Thus, such destinations:
  - Cannot use real-time apply
  - Cannot be configured as a redo forwarding destination
- The actual number of archiver processes in use at any given time may vary based on the archiver workload and the value of the LOG_ARCHIVE_MAX_PROCESSES initialization parameter. For example, if the total of MAX_CONNECTIONS attributes on all destinations exceeds the value of LOG_ARCHIVE_MAX_PROCESSES, then Data Guard will use as many ARC*n* processes as possible, but the number may be less than what is specified by the MAX_CONNECTIONS attribute.
- When using multiple ARC*n* processes in an Oracle RAC environment, configure the primary instance to transport redo data to a single standby database instance. If redo transport services are not configured as such, then archival will return to the default behavior for remote archival, which is to transport redo data using a single ARC*n* process.

Examples

The following example shows the MAX_CONNECTIONS attribute:

```
LOG_ARCHIVE_DEST_1='LOCATION=/arch/dest'
```

```
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_3='SERVICE=denver MAX_CONNECTIONS=3'
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

MAX_FAILURE

Controls the consecutive number of times redo transport services attempt to reestablish communication and transmit redo data to a failed destination before the primary database gives up on the destination.

Category	MAX_FAILURE=*count*
Data type	Numeric
Valid value	>=0
Default value	None
Requires attributes	REOPEN
Conflicts with attributes	None
Corresponds to	MAX_FAILURE, FAILURE_COUNT, and REOPEN_SECS columns of the V$ARCHIVE_DEST view

Usage Notes

- The MAX_FAILURE attribute is optional. By default, there are an unlimited number of archival attempts to the failed destination.
- This attribute is useful for providing failure resolution for destinations to which you want to retry transmitting redo data after a failure, but not retry indefinitely.
- When you specify the MAX_FAILURE attribute, you must also set the REOPEN attribute. Once the specified number of consecutive attempts is exceeded, the destination is treated as if the REOPEN attribute was not specified.
- You can view the failure count in the FAILURE_COUNT column of the V$ARCHIVE_DEST fixed view. The related column REOPEN_SECS identifies the REOPEN attribute value.

> **Note:** Once the failure count for the destination reaches the specified MAX_FAILURE attribute value, the only way to reuse the destination is to modify the MAX_FAILURE attribute value or any attribute. This has the effect of resetting the failure count to zero (0).

- The failure count is reset to zero (0) whenever the destination is modified by an ALTER SYSTEM SET statement. This avoids the problem of setting the MAX_FAILURE attribute to a value less than the current failure count value.
- Once the failure count is greater than or equal to the value set for the MAX_FAILURE attribute, the REOPEN attribute value is implicitly set to zero, which causes redo transport services to transport redo data to an alternate destination (defined with the ALTERNATE attribute) on the next archival operation.
- Redo transport services attempt to archive to the failed destination indefinitely if you do not specify the MAX_FAILURE attribute (or if you specify MAX_FAILURE=0), and you specify a nonzero value for the REOPEN attribute. If the destination has the MANDATORY attribute, the online redo log file is not reusable until it has been archived to this destination.

Examples

The following example allows redo transport services up to three consecutive archival attempts, tried every 5 seconds, to the `arc_dest` destination. If the archival operation fails after the third attempt, the destination is treated as if the `REOPEN` attribute was not specified.

```
LOG_ARCHIVE_DEST_1='LOCATION=/arc_dest REOPEN=5 MAX_FAILURE=3'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
```

NET_TIMEOUT

Specifies the number of seconds that the LGWR background process will block waiting for a redo transport destination to acknowledge redo data sent to it. If an acknowledgement is not received within NET_TIMEOUT seconds, an error is logged and the redo transport session to that destination is terminated.

Category	NET_TIMEOUT=*seconds*
Data type	Numeric
Valid values	1[1] to 1200
Default value	30 seconds
Requires attributes	SYNC
Conflicts with attributes	ASYNC (If you specify the ASYNC attribute, redo transport services ignores it; no error is returned.)
Corresponds to	NET_TIMEOUT column of the V$ARCHIVE_DEST view of the primary database

[1] Although a minimum value of 1 second is allowed, Oracle recommends a minimum value of 8 to 10 seconds to avoid disconnecting from the standby database due to transient network errors.

Usage Notes

- The NET_TIMEOUT attribute is optional. However, if you do not specify the NET_TIMEOUT attribute it will be set to 30 seconds, but the primary database can potentially stall. To avoid this situation, specify a small, nonzero value for the NET_TIMEOUT attribute so the primary database can continue operation after the user-specified timeout interval expires when waiting for status from the network server.

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 SYNC NET_TIMEOUT=40'
LOG_ARCHIVE_DEST_STATE_2=ENABLE
```

NOREGISTER

Indicates that the location of the archived redo log file should not be recorded at the corresponding destination.

Category	NOREGISTER
Data type	Keyword
Valid values	Not applicable
Default value	Not applicable
Requires attributes	SERVICE
Conflicts with attributes	LOCATION
Corresponds to	DESTINATION and TARGET columns of the V$ARCHIVE_DEST view

Usage Notes

- The NOREGISTER attribute *is optional* if the standby database destination is a part of a Data Guard configuration.
- The NOREGISTER attribute *is required* if the destination is not part of a Data Guard configuration.
- This attribute pertains to remote destinations only. The location of each archived redo log file is always recorded in the primary database control file.

Examples

The following example shows the NOREGISTER attribute:

```
LOG_ARCHIVE_DEST_5='NOREGISTER'
```

REOPEN

Specifies the minimum number of seconds before redo transport services should try to reopen a failed destination.

Category	REOPEN [=*seconds*]
Data Type	Numeric
Valid values	>=0 seconds
Default Value	300 seconds
Requires attributes	None
Conflicts with attributes	`Not applicable`
Corresponds to	`REOPEN_SECS` and `MAX_FAILURE` columns of the `V$ARCHIVE_DEST` view

Usage Notes

- The `REOPEN` attribute is optional.
- Redo transport services attempt to reopen failed destinations at log switch time.
- Redo transport services check if the time of the last error plus the `REOPEN` interval is less than the current time. If it is, redo transport services attempt to reopen the destination.
- `REOPEN` applies to all errors, not just connection failures. These errors include, but are not limited to, network failures, disk errors, and quota exceptions.
- If you specify `REOPEN` for an optional destination, it is possible for the Oracle database to overwrite online redo log files if there is an error. If you specify `REOPEN` for a `MANDATORY` destination, redo transport services will stall the primary database when it is not possible to successfully transmit redo data. When this situation occurs, consider the following options:
  - Change the destination by deferring the destination, specifying the destination as optional, or changing the `SERVICE` attribute value.
  - Specify an alternate destination.
  - Disable the destination.

Examples

The following example shows the `REOPEN` attribute.

```
LOG_ARCHIVE_DEST_3='SERVICE=stby1 MANDATORY REOPEN=60'
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

SYNC and ASYNC

Specifies whether the synchronous (SYNC) or asynchronous (ASYNC) redo transport mode is to be used.

Category	SYNC	ASYNC
Data type	Keyword	Keyword
Valid values	Not applicable	Not applicable
Default value	Not applicable	None
Requires attributes	None	None
Conflicts with attributes	ASYNC, LOCATION	SYNC, LOCATION
Corresponds to	TRANSMIT_MODE column of the V$ARCHIVE_DEST view	TRANSMIT_MODE and column of the V$ARCHIVE_DEST view

Usage Notes

- The redo data generated by a transaction must have been received by every enabled destination that has the SYNC attribute before that transaction can commit.
- The redo data generated by a transaction need *not* have been received at a destination that has the ASYNC attribute before that transaction can commit. This is the default behavior if neither SYNC or ASYNC is specified.

Examples

The following example shows the SYNC attribute with the LOG_ARCHIVE_DEST_*n* parameter.

```
LOG_ARCHIVE_DEST_3='SERVICE=stby1 SYNC'
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

VALID_FOR

Specifies whether redo data will be written to a destination, based on the following factors:

- Whether the database is *currently* running in the primary or the standby role
- Whether online redo log files, standby redo log files, or both are *currently* being archived on the database at this destination

Category	VALID_FOR=(*redo_log_type*, *database_role*)
Data Type	String value
Valid values	Not applicable
Default Value	VALID_FOR=(ALL_LOGFILES, ALL_ROLES)
Requires attributes	None
Conflicts with attributes	None
Corresponds to	VALID_NOW, VALID_TYPE, and VALID_ROLE columns in the V$ARCHIVE_DEST view

Usage Notes

- The VALID_FOR attribute is optional. However, Oracle recommends that the VALID_FOR attribute be specified for each redo transport destination at each database in a Data Guard configuration so that redo transport continues after a role transition to any standby database in the configuration.
- To configure these factors for each LOG_ARCHIVE_DEST_*n* destination, you specify this attribute with a pair of keywords: VALID_FOR=(*redo_log_type*, *database_role*):
  - The *redo_log_type* keyword identifies the destination as valid for archiving one of the following:
    - ONLINE_LOGFILE—This destination is valid only when archiving online redo log files.
    - STANDBY_LOGFILE—This destination is valid only when archiving standby redo log files.
    - ALL_LOGFILES— This destination is valid when archiving either online redo log files or standby redo log files.
  - The *database_role* keyword identifies the role in which this destination is valid for archiving:
    - PRIMARY_ROLE—This destination is valid only when the database is running in the primary role.
    - STANDBY_ROLE—This destination is valid only when the database is running in the standby role.
    - ALL_ROLES—This destination is valid when the database is running in either the primary or the standby role.
- 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.

- The VALID_FOR attribute enables you to use the same initialization parameter file for both the primary and standby roles.

Example

The following example shows the default VALID_FOR keyword pair:

```
LOG_ARCHIVE_DEST_1='LOCATION=/disk1/oracle/oradata VALID_FOR=(ALL_LOGFILES, ALL_ROLES)'
```

When this database is running in either the primary or standby role, destination 1 archives all log files to the /disk1/oracle/oradata local directory location.

16

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 Language Reference* for complete syntax and descriptions about these and other SQL statements.

See Chapter 14 for a list of initialization parameters that you can set and dynamically update using the `ALTER SYSTEM SET` statement.

16.1 ALTER DATABASE Statements

Table 16–1 describes `ALTER DATABASE` statements that are relevant to Data Guard.

Table 16–1 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 9.3.5 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.
`[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 *Oracle Database SQL Language Reference* for more information.

Table 16–1 (Cont.) ALTER DATABASE Statements Used in Data Guard Environments

ALTER DATABASE Statement	Description
`COMMIT TO SWITCHOVER TO [[PRIMARY] | [[PHYSICAL|LOGICAL] [STANDBY]] [WITH | WITHOUT] SESSION SHUTDOWN] [WAIT | NOWAIT]`	Performs a switchover to: ▪ 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 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 8.2.1 and Section 8.3.1 for examples of this statement.
`CONVERT TO [[PHYSICAL|SNAPSHOT] STANDBY] DATABASE`	Converts a physical standby database into a snapshot standby database and vice versa.
`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 for an example of this statement.
`DROP [STANDBY] LOGFILE` *logfile_descriptor*	Drops all members of an online redo log file group or standby redo log file group. See Section 9.3.5 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 to prevent inconsistent standby databases.: 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.
`GUARD`	Controls user access to tables in a logical standby database. Possible values are `ALL`, `STANDBY`, and `NONE`. See Section 10.2 for more information.
`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.
`PREPARE TO SWITCHOVER TO [PRIMARY] | [[PHYSICAL|LOGICAL] [STANDBY]] [WITH | WITHOUT] SESSION SHUTDOWN] [WAIT | NOWAIT]`	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 8.3.1 for examples of this statements.

Table 16–1 (Cont.) ALTER DATABASE Statements Used in Data Guard Environments

ALTER DATABASE Statement	Description
`RECOVER MANAGED STANDBY DATABASE [ { DISCONNECT [FROM SESSION] | USING CURRENT LOGFILE | NODELAY | UNTIL CHANGE integer }...]`	This statement starts and controls Redo Apply on physical standby databases. You can use the `RECOVER MANAGED STANDBY DATABASE` clause on a physical standby database that is mounted, open, or closed. See Step 4 in Section 3.2.6 and Section 7.3 for examples. **Note:** Several clauses and keywords were deprecated and are supported for backward compatibility only. See *Oracle Database SQL Language Reference* for more information about these clauses.
`RECOVER MANAGED STANDBY DATABASE CANCEL`	The `CANCEL` clause cancels Redo Apply on a physical standby database after applying the current archived redo log file. **Note:** Several clauses and keywords were deprecated and are supported for backward compatibility only. See *Oracle Database SQL Language Reference* for more information about these clauses.
`RECOVER MANAGED STANDBY DATABASE FINISH`	The `FINISH` clause initiates failover on the target physical standby database and recovers the current standby redo log files. Use the `FINISH` clause only in the event of the failure of the primary database. This clause overrides any delay intervals specified. See Step 4 in Section 8.2.2 for examples. **Note:** Several clauses and keywords were deprecated and are supported for backward compatibility only. See *Oracle Database SQL Language Reference* for more information about these clauses.
`REGISTER [OR REPLACE] [PHYSICAL|LOGICAL] LOGFILE` *`filespec`*	Allows the registration of manually archived redo log files.
`RECOVER TO LOGICAL STANDBY` *new_database_name*	Instructs apply services to continue applying changes to the *physical* standby database until you issue the command to convert the database to a *logical* standby database. See Section 4.2.4.1 for more information.
`RESET DATABASE TO INCARNATION` *integer*	Resets the target recovery incarnation for the database from the current incarnation to a different incarnation.
`SET STANDBY DATABASE TO MAXIMIZE {PROTECTION|AVAILABILITY|PERFORMANCE}`	Use this clause to specify the level of protection for the data in your Data Guard configuration. You specify this clause from the primary database, which must be mounted but not open.

Table 16–1 (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 7.4.1 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 8.3.2 for an example of this statement.
`ACTIVATE [PHYSICAL|LOGICAL] STANDBY DATABASE FINISH APPLY]`	Performs a failover. The standby database must be mounted before it can be activated with this statement. **Note:** Do not use the `ALTER DATABASE ACTIVATE STANDBY DATABASE` statement to failover because it causes data loss. Instead, use the following best practices: ▪ For physical standby databases, use the `ALTER DATABASE RECOVER MANAGED STANDBY DATABASE` statement with the `FINISH` keyword to perform the role transition as quickly as possible with little or no data loss and without rendering other standby databases unusable. **Note:** The failover operation adds an end-of-redo marker to the header of the last log file being archived and sends the redo to all enabled destinations that are valid for the primary role (specified with the `VALID_FOR=(PRIMARY_ROLE, *_LOGFILES)` or the `VALID_FOR=(ALL_ROLES, *_LOGFILES)` attributes). ▪ For logical standby databases, use the `ALTER DATABASE PREPARE TO SWITCHOVER` and `ALTER DATABASE COMMIT TO SWITCHOVER` statements.

16.2 ALTER SESSION Statements

Table 16–2 describes an `ALTER SESSION` statement that is relevant to Data Guard.

Table 16–2 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 10.5.4 for more information.

17

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 17–1 describes the views and indicates if a view applies to physical standby databases, logical standby databases, snapshot standby databases, or primary databases. See *Oracle Database Reference* for complete information about views.

Table 17–1 Views That Are Pertinent to Data Guard Configurations

View	Database	Description
DBA_LOGSTDBY_EVENTS	Logical only	Contains information about the activity of a logical standby database. It can be used to determine the cause of failures that occur when SQL Apply is applying redo to a logical standby database.
DBA_LOGSTDBY_HISTORY	Logical only	Displays the history of switchovers and failovers for logical standby databases in a Data Guard configuration. It does this by showing the complete sequence of redo log streams processed or created on the local system, across all role transitions. (After a role transition, a new log stream is started and the log stream sequence number is incremented by the new primary database.)
DBA_LOGSTDBY_LOG	Logical only	Shows the log files registered for logical standby databases.
DBA_LOGSTDBY_NOT_UNIQUE	Logical only	Identifies tables that have no primary and no non-null unique indexes.
DBA_LOGSTDBY_PARAMETERS	Logical only	Contains the list of parameters used by SQL Apply.
DBA_LOGSTDBY_SKIP	Logical only	Lists the tables that will be skipped by SQL Apply.
DBA_LOGSTDBY_SKIP_TRANSACTION	Logical only	Lists the skip settings chosen.
DBA_LOGSTDBY_UNSUPPORTED	Logical only	Identifies the schemas and tables (and columns in those tables) that contain unsupported data types. Use this view when you are preparing to create a logical standby database.
V$ARCHIVE_DEST	Primary, physical, snapshot, and logical	Describes all of the destinations in the Data Guard configuration, including each destination's current value, mode, and status. **Note:** The information in this view does not persist across an instance shutdown.
V$ARCHIVE_DEST_STATUS	Primary, physical, snapshot, and logical	Displays runtime and configuration information for the archived redo log destinations. **Note:** The information in this view does not persist across an instance shutdown.

Table 17–1 (Cont.) Views That Are Pertinent to Data Guard Configurations

View	Database	Description
V$ARCHIVE_GAP	Physical, snapshot, and logical	Displays information to help you identify a gap in the archived redo log files.
V$ARCHIVED_LOG	Primary, physical, snapshot, and logical	Displays archive redo log information from the control file, including names of the archived redo log files.
V$DATABASE	Primary, physical, snapshot, and logical	Provides database information from the control file. Includes information about fast-start failover (available only with the Data Guard broker).
V$DATABASE_INCARNATION	Primary, physical, snapshot, 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, snapshot, and logical	Provides datafile information from the control file.
V$DATAGUARD_CONFIG	Primary, physical, snapshot, and logical	Lists the unique database names defined with the DB_UNIQUE_NAME and LOG_ARCHIVE_CONFIG initialization parameters.
V$DATAGUARD_STATS	Primary, physical, snapshot, and logical	Displays how much redo data generated by the primary database is not yet available on the standby database, showing how much redo data could be lost if the primary database were to crash at the time you queried this view. You can query this view on any instance of a standby database in a Data Guard configuration. If you query this view on a primary database, then the column values are cleared. See also Section 8.1.2 on page 8-2 for an example and more information.
V$DATAGUARD_STATUS	Primary, physical, snapshot, and logical	Displays and records events that would typically be triggered by any message to the alert log or server process trace files.
V$FS_FAILOVER_STATS	Primary	Displays statistics about fast-start failover occurring on the system.
V$LOG	Primary, physical, snapshot, and logical	Contains log file information from the online redo log files.
V$LOGFILE	Primary, physical, snapshot, and logical	Contains information about the online redo log files and standby redo log files.
V$LOG_HISTORY	Primary, physical, snapshot, and logical	Contains log history information from the control file.

Table 17–1 (Cont.) Views That Are Pertinent to Data Guard Configurations

View	Database	Description
V$LOGSTDBY_PROCESS	Logical only	Provides dynamic information about what is happening with SQL Apply. This view is very helpful when you are diagnosing performance problems during SQL Apply on the logical standby database, and it can be helpful for other problems.
V$LOGSTDBY_PROGRESS	Logical only	Displays the progress of SQL Apply on the logical standby database.
V$LOGSTDBY_STATE	Logical only	Consolidates information from the V$LOGSTDBY_PROCESS and V$LOGSTDBY_STATS views about the running state of SQL Apply and the logical standby database.
V$LOGSTDBY_STATS	Logical only	Displays LogMiner statistics, current state, and status information for a logical standby database during SQL Apply. If SQL Apply is not running, the values for the statistics are cleared.
V$LOGSTDBY_TRANSACTION	Logical only	Displays information about all active transactions being processed by SQL Apply on the logical standby database.
V$MANAGED_STANDBY	Physical and snapshot	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$REDO_DEST_RESP_HISTOGRAM	Primary	Contains the response time information for destinations that are configured for SYNC transport.
V$STANDBY_LOG	Physical, snapshot, and logical	Contains log file information from the standby redo log files.

Part III

Appendixes

This part contains the following appendixes:

- Appendix A, "Troubleshooting Data Guard"
- Appendix B, "Upgrading Databases in a Data Guard Configuration"
- Appendix C, "Data Type and DDL Support on a Logical Standby Database"
- Appendix D, "Data Guard and Oracle Real Application Clusters"
- Appendix E, "Cascaded Destinations"
- Appendix F, "Creating a Standby Database with Recovery Manager"
- Appendix G, "Setting Archive Tracing"

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:

A.1.1 Renaming Datafiles with the ALTER DATABASE Statement

You cannot rename the datafile on the standby site when the `STANDBY_FILE_MANAGEMENT` initialization parameter is set to `AUTO`. When you set the `STANDBY_FILE_MANAGEMENT` initialization parameter to `AUTO`, use of the following SQL statements is not allowed:

- `ALTER DATABASE RENAME`
- `ALTER DATABASE ADD/DROP LOGFILE`
- `ALTER DATABASE ADD/DROP STANDBY LOGFILE MEMBER`
- `ALTER DATABASE CREATE DATAFILE AS`

If you attempt to use any of these statements on the standby database, an error is returned. For example:

```
SQL> ALTER DATABASE RENAME FILE '/disk1/oracle/oradata/payroll/t_db2.log' to 'dummy';
alter database rename file '/disk1/oracle/oradata/payroll/t_db2.log' to 'dummy'
*
ERROR at line 1:
ORA-01511: error in renaming log/datafiles
ORA-01270: RENAME operation is not allowed if STANDBY_FILE_MANAGEMENT is auto
```

See Section 9.3.1 to learn how to add datafiles to a physical standby database.

A.1.2 Standby Database Does Not Receive Redo Data from the Primary Database

If the standby site is not receiving redo data, query the V$ARCHIVE_DEST view and check for error messages. For example, enter the following query:

```
SQL> SELECT DEST_ID "ID",
  2> STATUS "DB_status",
  3> DESTINATION "Archive_dest",
  4> ERROR "Error"
  5> FROM V$ARCHIVE_DEST WHERE DEST_ID <=5;

ID DB_status Archive_dest                  Error
-- --------- ------------------------------ ------------------------------------
 1  VALID    /vobs/oracle/work/arc_dest/arc
 2  ERROR    standby1                       ORA-16012: Archivelog standby database identifier mismatch
 3  INACTIVE
 4  INACTIVE
 5  INACTIVE
5 rows selected.
```

If the output of the query does not help you, check the following list of possible issues. If any of the following conditions exist, redo transport services will fail to transmit redo data to the standby database:

- The service name for the standby instance is not configured correctly in the tnsnames.ora file for the primary database.
- The Oracle Net service name specified by the LOG_ARCHIVE_DEST_*n* parameter for the primary database is incorrect.
- The LOG_ARCHIVE_DEST_STATE_*n* parameter for the standby database is not set to the value ENABLE.
- The listener.ora file has not been configured correctly for the standby database.
- The listener is not started at the standby site.
- The standby instance is not started.
- You have added a standby archiving destination to the primary SPFILE or text initialization parameter file, but have not yet enabled the change.
- Redo transport authentication has not been configured properly. See section 3.1.2 for redo transport authentication configuration requirements.
- 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.1.3 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 a MANDATORY destination, redo transport services stall the primary database when redo data cannot be successfully transmitted.

The REOPEN attribute is required when you use the MAX_FAILURE attribute. Example A–1 shows how to set a retry time of 5 seconds and limit retries to 3 times.

Example A–1 Setting a Retry Time and Limit

```
LOG_ARCHIVE_DEST_1='LOCATION=/arc_dest REOPEN=5 MAX_FAILURE=3'
```

Use the ALTERNATE attribute of the LOG_ARCHIVE_DEST_*n* parameter to specify alternate archive destinations. An alternate archiving destination can be used when the transmission of redo data to a standby database fails. If transmission fails and the REOPEN attribute was not specified or the MAX_FAILURE attribute threshold was exceeded, redo transport services attempts to transmit redo data to the alternate destination on the next archival operation.

Use the NOALTERNATE attribute to prevent the original archive destination from automatically changing to an alternate archive destination when the original archive destination fails.

Example A–2 shows how to set the initialization parameters so that a single, mandatory, local destination will automatically fail over to a different destination if any error occurs.

Example A–2 Specifying an Alternate Destination

```
LOG_ARCHIVE_DEST_1='LOCATION=/disk1 MANDATORY ALTERNATE=LOG_ARCHIVE_DEST_2'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_2='LOCATION=/disk2 MANDATORY'
LOG_ARCHIVE_DEST_STATE_2=ALTERNATE
```

If the LOG_ARCHIVE_DEST_1 destination fails, the archiving process will automatically switch to the LOG_ARCHIVE_DEST_2 destination at the next log file switch on the primary database.

A.3 Handling Logical Standby Database Failures

An important tool for handling logical standby database failures is the DBMS_LOGSTDBY.SKIP_ERROR procedure. Depending on how important a table is, you might want to do one of the following:

- Ignore failures for a table or specific DDL

- Associate a stored procedure with a filter so at runtime a determination can be made about skipping the statement, executing this statement, or executing a replacement statement

Taking one of these actions prevents SQL Apply from stopping. Later, you can query the DBA_LOGSTDBY_EVENTS view to find and correct any problems that exist. See *Oracle Database PL/SQL Packages and Types Reference* for more information about using the DBMS_LOGSTDBY package with PL/SQL callout procedures.

A.4 Problems Switching Over to a Physical Standby Database

In most cases, following the steps described in Chapter 8 will result in a successful switchover. However, if the switchover is unsuccessful, the following sections may help you to resolve the problem:

- Switchover Fails Because Redo Data Was Not Transmitted
- Switchover Fails Because SQL Sessions Are Still Active
- Switchover Fails Because User Sessions Are Still Active
- Switchover Fails with the ORA-01102 Error
- Redo Data Is Not Applied After Switchover
- Roll Back After Unsuccessful Switchover and Start Over

A.4.1 Switchover Fails Because Redo Data Was Not Transmitted

If the switchover does not complete successfully, you can query the SEQUENCE# column in the V$ARCHIVED_LOG view to see if the last redo data transmitted from the original primary database was applied on the standby database. If the last redo data was not transmitted to the standby database, you can manually copy the archived redo log file containing the redo data from the original primary database to the old standby database and register it with the SQL ALTER DATABASE REGISTER LOGFILE *file_specification* statement. If you then start apply services, the archived redo log file will be applied automatically. Query the SWITCHOVER_STATUS column in the V$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 17 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 8.2.1 for physical standby databases or Section 8.3.1 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.

Table A–1 (Cont.) Common Processes That Prevent Switchover

Type of Process	Process Description	Corrective Action
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 emctl stop agent 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 see which sessions are still active, query the V$SESSION view:

```
SQL> SELECT SID, PROCESS, PROGRAM FROM V$SESSION;
```

A.4.4 Switchover Fails with the ORA-01102 Error

Suppose the standby database and the primary database reside on the same site. After both the ALTER DATABASE COMMIT TO SWITCHOVER TO PHYSICAL STANDBY and the ALTER DATABASE COMMIT TO SWITCHOVER TO PRIMARY statements are successfully executed, shut down and restart the physical standby database and the primary database.

Note: It is not necessary to shut down and restart the physical standby database if it has not been opened read-only since the instance was started.

However, the startup of the second database fails with ORA-01102 error "cannot mount database in EXCLUSIVE mode."

This could happen during the switchover if you did not set the DB_UNIQUE_NAME parameter in the initialization parameter file that is used by the standby database (that is, the original primary database). If the DB_UNIQUE_NAME parameter of the standby database is not set, the standby and the primary databases both use the same mount lock and cause the ORA-01102 error during the startup of the second database.

Action: Add DB_UNIQUE_NAME=*unique_database_name* to the initialization parameter file used by the standby database, and shut down and restart the standby and primary databases.

A.4.5 Redo Data Is Not Applied After Switchover

The archived redo log files are not applied to the new standby database after the switchover.

This might happen because some environment or initialization parameters were not properly set after the switchover.

Action:

- Check the `tnsnames.ora` file at the new primary site and the `listener.ora` file at the new standby site. There should be entries for a listener at the standby site and a corresponding service name at the primary site.
- Start the listener at the standby site if it has not been started.
- Check if the `LOG_ARCHIVE_DEST_`*n* initialization parameter was set to properly transmit redo data from the primary site to the standby site. For example, query the `V$ARCHIVE_DEST` fixed view at the primary site as follows:

  ```
  SQL> SELECT DEST_ID, STATUS, DESTINATION FROM V$ARCHIVE_DEST;
  ```

  If you do not see an entry corresponding to the standby site, you need to set `LOG_ARCHIVE_DEST_`*n* and `LOG_ARCHIVE_DEST_STATE_`*n* initialization parameters.
- Set the `STANDBY_ARCHIVE_DEST` and `LOG_ARCHIVE_FORMAT` initialization parameters correctly at the standby site so that the archived redo log files are applied to the desired location. (Note that the `STANDBY_ARCHIVE_DEST` parameter has been deprecated and is supported for backward compatibility only.)
- At the standby site, set the `DB_FILE_NAME_CONVERT` and `LOG_FILE_NAME_CONVERT` initialization parameters. Set the `STANDBY_FILE_MANAGEMENT` initialization parameter to `AUTO` if you want the standby site to automatically add new datafiles that are created at the primary site.

A.4.6 Roll Back After Unsuccessful Switchover and Start Over

For physical standby databases in situations where an error occurred and it is not possible to continue with the switchover, it might still be possible to revert the new physical standby database back to the primary role by using the following steps:

1. Connect to the new standby database (old primary), and issue the following statement to convert it back to the primary role:

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

   If this statement is successful, then shut down (if necessary) and restart the database. Once restarted, the database will be running in the primary database role, and you do not need to perform any more steps.

   If this statement is unsuccessful, then continue with Step 3.
2. When the switchover to change the role from primary to physical standby was initiated, a trace file was written in the log directory. This trace file contains the SQL statements required to re-create the original primary control file. Locate the trace file and extract the SQL statements into a temporary file. Execute the temporary file from SQL*Plus. This will revert the new standby database back to the primary role.
3. Shut down the original physical standby database.
4. Create a new standby control file. This is necessary to resynchronize the primary database and physical standby database. Copy the physical standby control file to the original physical standby system. Section 3.2.2 describes how to create a physical standby control file.
5. Restart the original physical standby instance.

   If this procedure is successful and archive gap management is enabled, the FAL processes will start and re-archive any missing archived redo log files to the physical standby database. Force a log switch on the primary database and

examine the alert logs on both the primary database and physical standby database to ensure the archived redo log file sequence numbers are correct.

See Section 6.3.3.1 for information about archive gap management and Appendix G for information about locating the trace files.

6. Try the switchover again.

   At this point, the Data Guard configuration has been rolled back to its initial state, and you can try the switchover operation again (after correcting any problems that might have led to the initial unsuccessful switchover).

A.5 Problems Switching Over to a Logical Standby Database

A switchover operation involving a logical standby database usually consists of two phases: preparing and committing. The exceptions to this are for rolling upgrades of Oracle software using a logical standby database or if you are using Data Guard broker. If you experience failures in the context of doing a rolling upgrade using a logical standby database or during a switchover operation initiated by Data Guard broker, you should go directly to Section A.5.2.

Note: Oracle recommends that Flashback Database be enabled for all databases in a Data Guard configuration. The steps in this section assume that you have Flashback Database enabled on all databases in your Data Guard configuration.

A.5.1 Failures During the Prepare Phase of a Switchover Operation

If a failure occurs during the preparation phase of a switchover operation, you should cancel the switchover and retry the switchover operation from the very beginning.

A.5.1.1 Failure While Preparing the Primary Database

If you encounter failure while executing the `ALTER DATABASE PREPARE TO SWITCHOVER TO LOGICAL STANDBY` statement, you can cancel the prepare phase of a switchover by issuing the following SQL statement at the primary database:

```
SQL> ALTER DATABASE PREPARE TO SWITCHOVER TO LOGICAL STANDBY CANCEL;
```

You can now retry the switchover operation from the beginning.

A.5.1.2 Failure While Preparing the Logical Standby Database

If you encounter failure while executing the `ALTER DATABASE PREPARE TO SWITCHOVER TO PRIMARY` statement, you will need to cancel the prepare operation at the primary database and at the target standby database. Take the following steps:

1. At the primary database, cancel the statement you had issued to prepare for the switchover:

   ```
   SQL> ALTER DATABASE PREPARE TO SWITCHOVER TO LOGICAL STANDBY CANCEL;
   ```

2. At the logical standby database that was the target of the switchover, cancel the statement you had issued to prepare to switch over:

   ```
   SQL> ALTER DATABASE PREPARE TO SWITCHOVER TO PRIMARY CANCEL;
   ```

   You can now retry the switchover operation from the beginning.

A.5.2 Failures During the Commit Phase of a Switchover Operation

Although committing to a switchover involves a single SQL statement, internally a number of operations are performed. The corrective actions that you need to take depend on the state of the commit to switchover operation when the error was encountered.

A.5.2.1 Failure to Convert the Original Primary Database

If you encounter failures while executing the `ALTER DATABASE COMMIT TO SWITCHOVER TO LOGICAL STANDBY` statement, you can take the following steps:

1. Check the `DATABASE_ROLE` column of the `V$DATABASE` fixed view on the original primary database:

   ```
   SQL> SELECT DATABASE_ROLE FROM V$DATABASE;
   ```

   - If the column contains a value of `LOGICAL STANDBY`, the switchover operation has completed, but has failed during a post-switchover task. In this situation, Oracle recommends that you shut down and reopen the database.
   - If the column contains a value of `PRIMARY`, proceed to Step 2.

2. Perform the following query on the original primary:

   ```
   SQL> SELECT COUNT(*) FROM SYSTEM.LOGSTDBY$PARAMETERS
     2> WHERE NAME = 'END_PRIMARY';
   ```

   - If the query returns a 0, the primary is in a state identical to that it was in before the commit to switchover command was issued. You do not need to take any corrective action. You can proceed with the commit to switchover operation or cancel the switchover operation as outlined in Section A.5.1.2.
   - If the query returns a 1, the primary is in an inconsistent state, and you need to proceed to Step 3.

3. Take corrective action at the original primary database to maintain its ability to be protected by existing or newly instantiated logical standby databases.

   You can either fix the underlying cause of the error raised during the commit to switchover operation and reissue the SQL statement (`ALTER DTABASE COMMIT TO SWITCHOVER TO LOGICAL STANDBY`) or you can take the following steps:

   a. From the alert log of the instance where you initiated the commit to switchover command, determine the SCN needed to flash back to the original primary. This information is displayed after the `ALTER DATABASE COMMIT TO SWITCHOVER TO LOGICAL STANDBY` SQL statement:

      ```
      LOGSTDBY: Preparing the COMMIT TO SWITCHOVER TO LOGICAL STANDBY DDL at scn
      [flashback_scn].
      ```

   b. Shut down all instances of the primary database:

      ```
      SQL> SHUTDOWN IMMEDIATE;
      ```

   c. Mount the primary database in exclusive mode:

      ```
      SQL> STARTUP MOUNT;
      ```

   d. Flash back the database to the SCN taken from the alert log:

      ```
      SQL> FLASHBACK DATABASE TO BEFORE SCN <flashback_scn>;
      ```

   e. Open the primary database:

```
SQL> STARTUP;
```

f. Lower the database guard at the original primary database:

```
SQL> ALTER DATABASE GUARD NONE;
```

At this point the primary is in a state identical to that it was in before the commit switchover command was issued. You do not need to take any corrective action. you can proceed with the commit to switchover operation or cancel the switchover operation as outlined in Section A.5.1.1.

A.5.2.2 Failure to Convert the Target Logical Standby Database

If you encounter failures while executing the `ALTER DATABASE COMMIT TO SWITCHOVER TO PRIMARY` statement, take the following steps:

1. Check the `DATABASE_ROLE` column of the `V$DATABASE` fixed view on the target standby database:

   ```
   SQL> SELECT DATABASE_ROLE FROM V$DATABASE;
   ```

   - If the column contains a value `PRIMARY`, the switchover operation has completed, but has failed during a post-switchover task. In this situation, you must perform the following steps:

     a. Shut down and reopen the database.

     b. Issue an `ALTER DATABASE GUARD NONE` command to remove write restrictions to the database.

   - If the column contains a value of `LOGICAL STANDBY`, proceed to Step 2.

2. Perform the following query on the target logical standby:

   ```
   SQL> SELECT COUNT(*) FROM SYSTEM.LOGSTDBY$PARAMETERS
     2> WHERE NAME = 'BEGIN_PRIMARY';
   ```

   - If the query returns a 0, the logical standby is in a state identical to that it was in before the commit to switchover command was issued. You do not need to take any corrective action. You can proceed with the commit to switchover operations or cancel the switchover operation as outlined in Section A.5.1.2.
   - If the query returns a 1, the logical standby is in an inconsistent state, and you should proceed to Step 3.

3. Take corrective action at the logical standby to maintain its ability to either become the new primary or become a bystander to a different new primary.

   You can either fix the underlying cause of the error raised during the commit to switchover operation and reissue the SQL statement (`ALTER DATABASE COMMIT TO SWITCHOVER TO PRIMARY`) or you can take the following steps to flash back the logical standby database to a point of consistency just prior to the commit to switchover attempt:

   a. From the alert log of the instance where you initiated the commit to switchover command, determine the SCN needed to flash back to the logical standby. This information is displayed after the `ALTER DATABASE COMMIT TO SWITCHOVER TO PRIMARY` SQL statement:

   ```
   LOGSTDBY: Preparing the COMMIT TO SWITCHOVER TO PRIMARY DDL at scn
   [flashback_scn].
   ```

   b. Shut down all instances of the target standby database:

```
SQL> SHUTDOWN IMMEDIATE;
```

c. Mount the target logical standby database:

```
SQL> STARTUP MOUNT;
```

d. Flash back the target logical standby to the desired SCN:

```
SQL> FLASHBACK DATABASE TO BEFORE SCN <flashback_scn>;
```

e. Open the database (in case of an Oracle RAC, open all instances);

```
SQL> STARTUP OPEN;
```

At this point the target standby is in a state identical to that it was in before the commit to switchover command was issued. You do not need to take any further corrective action. You can proceed with the commit to switchover operation.

A.6 What to Do If SQL Apply Stops

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.
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 17 for information about querying the DBA_LOGSTDBY_EVENTS view to determine the cause of failures.

A.7 Network Tuning for Redo Data Transmission

For optimal performance, set the Oracle Net SDU parameter to 32 kilobytes in each Oracle Net connect descriptor used by redo transport services.

The following example shows a database initialization parameter file segment that defines a remote destination `netserv`:

```
LOG_ARCHIVE_DEST_3='SERVICE=netserv'
```

The following example shows the definition of that service name in the `tnsnames.ora` file:

```
netserv=(DESCRIPTION=(SDU=32768)(ADDRESS=(PROTOCOL=tcp)(HOST=host) (PORT=1521))
(CONNECT_DATA=(SERVICE_NAME=srvc)))
```

The following example shows the definition in the `listener.ora` file:

```
LISTENER=(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=tcp)
(HOST=host)(PORT=1521))))

SID_LIST_LISTENER=(SID_LIST=(SID_DESC=(SDU=32768)(SID_NAME=sid)
(GLOBALDBNAME=srvc)(ORACLE_HOME=/oracle)))
```

If you archive to a remote site using a high-latency or high-bandwidth network link, you can improve performance by using the `SQLNET.SEND_BUF_SIZE` and `SQLNET.RECV_BUF_SIZE` Oracle Net profile parameters to increase the size of the network send and receive I/O buffers.

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

A.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 setting 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 standby redo log files on each standby database exactly matches the size of the online redo log files on the primary database.

At log switch time, if there are no available standby redo log files that match the size of the new current online redo log file on the primary database:

- The primary database will shut down if it is operating in maximum protection mode,

  or

- The RFS process on the standby database will create an archived redo log file on the standby database and write the following message in the alert log:

  ```
  No standby log files of size <#> blocks available.
  ```

For example, if the primary database uses two online redo log groups whose log files are 100K, then the standby database should have 3 standby redo log groups with log file sizes of 100K.

Also, whenever you add a redo log group to the primary database, you must add a corresponding standby redo log group to the standby database. This reduces the probability that the primary database will be adversely affected because a standby redo log file of the required size is not available at log switch time.

A.10 Troubleshooting a Logical Standby Database

This section contains the following topics:

- Recovering from Errors
- Troubleshooting SQL*Loader Sessions
- Troubleshooting Long-Running Transactions
- Troubleshooting ORA-1403 Errors with Flashback Transactions

A.10.1 Recovering from Errors

Logical standby databases maintain user tables, sequences, and jobs. To maintain other objects, you must reissue the DDL statements seen in the redo data stream.

If SQL Apply fails, an error is recorded in the DBA_LOGSTDBY_EVENTS table. The following sections demonstrate how to recover from two such errors.

A.10.1.1 DDL Transactions Containing File Specifications

DDL statements are executed the same way on the primary database and the logical standby database. If the underlying file structure is the same on both databases, the DDL will execute on the standby database as expected.

If an error was caused by a DDL transaction containing a file specification that did not match in the logical standby database environment, perform the following steps to fix the problem:

1. Use the ALTER SESSION DISABLE GUARD statement to bypass the database guard so you can make modifications to the logical standby database:

```
SQL> ALTER SESSION DISABLE GUARD;
```

2. Execute the DDL statement, using the correct file specification, and then 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 IMMEDIATE
  2> SKIP FAILED TRANSACTION;
```

In some situations, the problem that caused the transaction to fail can be corrected and SQL Apply restarted without skipping the transaction. An example of this might be when available space is exhausted. (Do not let the primary and logical standby databases diverge when skipping DDL transactions. If possible, you should manually execute a compensating transaction in place of the skipped transaction.)

The following example shows SQL Apply stopping, the error being corrected, and then restarting SQL Apply:

```
SQL> SET LONG 1000
SQL> ALTER SESSION SET NLS_DATE_FORMAT  = 'DD-MON-YY HH24:MI:SS';

Session altered.

SQL> SELECT EVENT_TIME, COMMIT_SCN, EVENT, STATUS FROM DBA_LOGSTDBY_EVENTS;

EVENT_TIME              COMMIT_SCN
```

```
----------------- ---------------
EVENT
--------------------------------------------------------------------------------
STATUS
--------------------------------------------------------------------------------
22-OCT-03 15:47:58

ORA-16111: log mining and apply setting up

22-OCT-03 15:48:04          209627
insert into "SCOTT"."EMP"
values
   "EMPNO" = 7900,
   "ENAME" = 'ADAMS',
   "JOB" = 'CLERK',
   "MGR" IS NULL,
   "HIREDATE" = TO_DATE('22-OCT-03', 'DD-MON-RR'),
   "SAL" = 950,
   "COMM" IS NULL,
   "DEPTNO" IS NULL
ORA-01653: unable to extend table SCOTT.EMP by %200 bytes in tablespace T_TABLE
```

In the example, the `ORA-01653` message indicates that the tablespace was full and unable to extend itself. To correct the problem, add a new datafile to the tablespace. For example:

```
SQL> ALTER TABLESPACE t_table ADD DATAFILE '/dbs/t_db.f' SIZE 60M;
Tablespace altered.
```

Then, restart SQL Apply:

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

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

A.10.1.2 Recovering from DML Failures

Do not use the `SKIP_TRANSACTION` procedure to filter DML failures. Not only is the DML that is seen in the events table skipped, but so is all the DML associated with the transaction. This will cause multiple tables.

DML failures usually indicate a problem with a specific table. For example, assume the failure is an out-of-storage error that you cannot resolve immediately. The following steps demonstrate one way to respond to this problem.

1. Bypass the table, but not the transaction, by adding the table to the skip list:

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

   From this point on, DML activity for the `SCOTT.EMP` table is not applied. After you correct the storage problem, you can fix the table, provided you set up a database link to the primary database that has administrator privileges to run procedures in the `DBMS_LOGSTDBY` package.

2. Using the database link to the primary database, drop the local `SCOTT.EMP` table and then re-create it, and pull the data over to the standby database.

   ```
   SQL> ALTER DATABASE STOP LOGICAL STANDBY APPLY;
   SQL> EXECUTE DBMS_LOGSTDBY.INSTANTIATE_TABLE('SCOTT','EMP','PRIMARYDB');
   ```

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

3. To ensure a consistent view across the newly instantiated table and the rest of the database, wait for SQL Apply to catch up with the primary database before querying this table. Refer to Section 10.5.5, "Adding or Re-Creating Tables On a Logical Standby Database" for a detailed example.

A.10.2 Troubleshooting SQL*Loader Sessions

Oracle SQL*Loader provides a method of loading data from different sources into the Oracle Database. This section analyzes some of the features of the SQL*Loader utility as it pertains to SQL Apply.

Regardless of the method of data load chosen, the SQL*Loader control files contain an instruction on what to do to the current contents of the Oracle table into which the new data is to be loaded, via the keywords of APPEND and REPLACE. The following examples show how to use these keywords on a table named LOAD_STOK:

- When using the APPEND keyword, the new data to be loaded is appended to the contents of the LOAD_STOK table:

  ```
  LOAD DATA
  INTO TABLE LOAD_STOK APPEND
  ```

- When using the REPLACE keyword, the contents of the LOAD_STOK table are deleted prior to loading new data. Oracle SQL*Loader uses the DELETE statement to purge the contents of the table, in a single transaction:

  ```
  LOAD DATA
  INTO TABLE LOAD_STOK REPLACE
  ```

Rather than using the REPLACE keyword in the SQL*Loader script, Oracle recommends that prior to loading the data, issue the SQL*Plus TRUNCATE TABLE command against the table on the primary database. This will have the same effect of purging both the primary and standby databases copy of the table in a manner that is both fast and efficient because the TRUNCATE TABLE command is recorded in the online redo log files and is issued by SQL Apply on the logical standby database.

The SQL*Loader script may continue to contain the REPLACE keyword, but it will now attempt to DELETE zero rows from the object on the primary database. Because no rows were deleted from the primary database, there will be no redo recorded in the redo log files. Therefore, no DELETE statement will be issued against the logical standby database.

Issuing the REPLACE keyword without the SQL statement TRUNCATE TABLE provides the following potential problems for SQL Apply when the transaction needs to be applied to the logical standby database.

- If the table currently contains a significant number of rows, then these rows need to be deleted from the standby database. Because SQL Apply is not able to determine the original syntax of the statement, SQL Apply must issue a DELETE statement for each row purged from the primary database.

  For example, if the table on the primary database originally had 10,000 rows, then Oracle SQL*Loader will issue a single DELETE statement to purge the 10,000 rows. On the standby database, SQL Apply does not know that all rows are to be purged, and instead must issue 10,000 individual DELETE statements, with each statement purging a single row.

- If the table on the standby database does not contain an index that can be used by SQL Apply, then the `DELETE` statement will issue a Full Table Scan to purge the information.

  Continuing with the previous example, because SQL Apply has issued 10,000 individual `DELETE` statements, this could result in 10,000 Full Table Scans being issued against the standby database.

A.10.3 Troubleshooting Long-Running Transactions

One of the primary causes for long-running transactions in a SQL Apply environment is because of Full Table Scans. Additionally, long-running transactions could be the result of SQL statements being replicated to the standby database, such as when creating or rebuilding an index.

Identifying Long-Running Transactions

If SQL Apply is executing a single SQL statement for a long period of time, then a warning message similar to the following is reported in the alert log of the SQL Apply instance:

```
Mon Feb 17 14:40:15 2003
WARNING: the following transaction makes no progress
WARNING: in the last 30 seconds for the given message!
WARNING: xid =
0x0016.007.000017b6 cscn = 1550349, message# = 28, slavid = 1
knacrb: no offending session found (not ITL pressure)
```

Note the following about the warning message:

- This warning is similar to the warning message returned for interested transaction list (ITL) pressure, with the exception being the last line that begins with `knacrb`. The final line indicates:
  - A Full Table Scan may be occurring
  - This issue has nothing to do with interested transaction list (ITL) pressure
- This warning message is reported only if a single statement takes more than 30 seconds to execute.

It may not be possible to determine the SQL statement being executed by the long-running statement, but the following SQL statement may help in identifying the database objects on which SQL Apply is operating:

```
SQL> SELECT SAS.SERVER_ID
  2       , SS.OWNER
  3       , SS.OBJECT_NAME
  4       , SS.STATISTIC_NAME
  5       , SS.VALUE
  6    FROM V$SEGMENT_STATISTICS SS
  7       , V$LOCK L
  8       , V$STREAMS_APPLY_SERVER SAS
  9   WHERE SAS.SERVER_ID = &SLAVE_ID
 10     AND L.SID = SAS.SID
 11     AND L.TYPE = 'TM'
 12     AND SS.OBJ# = L.ID1;
```

Additionally, you can issue the following SQL statement to identify the SQL statement that has resulted in a large number of disk reads being issued per execution:

```
SQL> SELECT SUBSTR(SQL_TEXT,1,40)
```

```
  2       , DISK_READS
  3       , EXECUTIONS
  4       , DISK_READS/EXECUTIONS
  5       , HASH_VALUE
  6       , ADDRESS
  7    FROM V$SQLAREA
  8   WHERE DISK_READS/GREATEST(EXECUTIONS,1) > 1
  9     AND ROWNUM < 10
 10   ORDER BY DISK_READS/GREATEST(EXECUTIONS,1) DESC
```

Oracle recommends that all tables have primary key constraints defined, which automatically means that the column is defined as `NOT NULL`. For any table where a primary-key constraint cannot be defined, an index should be defined on an appropriate column that is defined as `NOT NULL`. If a suitable column does not exist on the table, then the table should be reviewed and, if possible, skipped by SQL Apply.

The following steps describe how to skip all DML statements issued against the `FTS` table on the `SCOTT` schema:

1. Stop SQL Apply:

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

2. Configure the skip procedure for the `SCOTT.FTS` table for all DML transactions:

```
SQL> EXECUTE DBMS_LOGSTDBY.SKIP(stmt => 'DML' , -
     schema_name => 'SCOTT' , -
     object_name => 'FTS');
PL/SQL procedure successfully completed
```

3. Start SQL Apply:

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

Troubleshooting ITL Pressure

Interested transaction list (ITL) pressure is reported in the alert log of the SQL Apply instance. Example A–3 shows an example of the warning messages.

Example A–3 Warning Messages Reported for ITL Pressure

```
Tue Apr 22 15:50:42 2003
WARNING: the following transaction makes no progress
WARNING: in the last 30 seconds for the given message!
WARNING: xid =
0x0006.005.000029fa cscn = 2152982, message# = 2, slavid = 17
```

Real-Time Analysis

The messages shown in Example A–3 indicate that the SQL Apply process (`slavid`) #17 has not made any progress in the last 30 seconds. To determine the SQL statement being issued by the Apply process, issue the following query:

```
SQL> SELECT SA.SQL_TEXT
  2    FROM V$SQLAREA SA
  3       , V$SESSION S
  4       , V$STREAMS_APPLY_SERVER SAS
  5   WHERE SAS.SERVER_ID = &SLAVEID
  6     AND S.SID = SAS.SID
  7   AND SA.ADDRESS = S.SQL_ADDRESS
SQL_TEXT
```

```
------------------------------------------------------------
insert into "APP"."LOAD_TAB_1" p("PK","TEXT")values(:1,:2)
```

An alternative method to identifying ITL pressure is to query the V$LOCK view, as shown in the following example. Any session that has a request value of 4 on a TX lock, is waiting for an ITL to become available.

```
SQL> SELECT SID,TYPE,ID1,ID2,LMODE,REQUEST
  2    FROM V$LOCK
  3   WHERE TYPE = 'TX'

SID        TY ID1        ID2        LMODE      REQUEST
---------- -- ---------- ---------- ---------- ----------
         8 TX     327688         48          6          0
        10 TX     327688         48          0          4
```

In this example, SID 10 is waiting for the TX lock held by SID 8.

Post-Incident Review

Pressure for a segment's ITL is unlikely to last for an extended period of time. In addition, ITL pressure that lasts for less than 30 seconds will not be reported in the standby databases alert log. Therefore, to determine which objects have been subjected to ITL pressure, issue the following statement:

```
SQL> SELECT SEGMENT_OWNER, SEGMENT_NAME, SEGMENT_TYPE
  2    FROM V$SEGMENT_STATISTICS
  3   WHERE STATISTIC_NAME = 'ITL WAITS'
  4     AND VALUE > 0
  5   ORDER BY VALUE
```

This statement reports all database segments that have had ITL pressure at some time since the instance was last started.

> **Note:** This SQL statement is not limited to a logical standby databases in the Data Guard environment. It is applicable to any Oracle database.

Resolving ITL Pressure

To increase the INITRANS integer for a particular database object, it is necessary to first stop SQL Apply.

> **See Also:** *Oracle Database SQL Language Reference* for more information about specifying the INITRANS integer, which it the initial number of concurrent transaction entries allocated within each data block allocated to the database object

The following example shows the necessary steps to increase the INITRANS for table load_tab_1 in the schema app.

1. Stop SQL Apply:

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

2. Temporarily bypass the database guard:

   ```
   SQL> ALTER SESSION DISABLE GUARD;
   Session altered.
   ```

3. Increase the INITRANS on the standby database. For example:

```
SQL> ALTER TABLE APP.LOAD_TAB_1 INITRANS 30;
Table altered
```

4. Reenable the database guard:

```
SQL> ALTER SESSION ENABLE GUARD;
Session altered
```

5. Start SQL Apply:

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

Also, consider modifying the database object on the primary database, so in the event of a switchover, the error should not occur on the new standby database.

A.10.4 Troubleshooting ORA-1403 Errors with Flashback Transactions

If SQL Apply returns the ORA-1403: No Data Found error, then it may be possible to use Flashback Transaction to reconstruct the missing data. This is reliant upon the UNDO_RETENTION initialization parameter specified on the standby database instance.

Under normal circumstances, the ORA-1403 error should not be seen in a logical standby database environment. The error occurs when data in a table that is being managed by SQL Apply is modified directly on the standby database and then the same data is modified on the primary database.

When the modified data is updated on the primary database and is subsequently received on the logical standby database, SQL Apply verifies the original version of the data is present on the standby database before updating the record. When this verification fails, the ORA-1403: No Data Found error is returned.

The Initial Error

When SQL Apply verification fails, the error message is reported in the alert log of the logical standby database and a record is inserted in the DBA_LOGSTDBY_EVENTS view.

The information in the alert log is truncated, while the error is reported in it's entirety in the database view. For example:

```
LOGSTDBY stmt: UPDATE "SCOTT"."MASTER"
  SET
    "NAME" = 'john'
  WHERE
    "PK" = 1 and
    "NAME" = 'andrew' and
    ROWID = 'AAAAAAAAEAAAAAPAAA'
LOGSTDBY status: ORA-01403: no data found
LOGSTDBY PID 1006, oracle@staco03 (P004)
LOGSTDBY XID 0x0006.00e.00000417, Thread 1, RBA 0x02dd.00002221.10
```

The Investigation

The first step is to analyze the historical data of the table that caused the error. This can be achieved using the VERSIONS clause of the SELECT statement. For example, you can issue the following query on the primary database:

```
SELECT VERSIONS_XID
```

```
      , VERSIONS_STARTSCN
      , VERSIONS_ENDSCN
      , VERSIONS_OPERATION
      , PK
      , NAME
   FROM SCOTT.MASTER
        VERSIONS BETWEEN SCN MINVALUE AND MAXVALUE
  WHERE PK = 1
  ORDER BY NVL(VERSIONS_STARTSCN,0);

VERSIONS_XID     VERSIONS_STARTSCN VERSIONS_ENDSCN V  PK NAME
---------------- ----------------- --------------- - --- -------
03001900EE070000           3492279         3492290 I   1 andrew
02000D00E4070000           3492290                 D   1 andrew
```

Depending upon the amount of undo retention that the database is configured to retain (`UNDO_RETENTION`) and the activity on the table, the information returned might be extensive and you may need to change the versions between syntax to restrict the amount of information returned.

From the information returned, it can be seen that the record was first inserted at SCN 3492279 and then was deleted at SCN 3492290 as part of transaction ID 02000D00E4070000.

Using the transaction ID, the database should be queried to find the scope of the transaction. This is achieved by querying the `FLASHBACK_TRANSACTION_QUERY` view.

```
SELECT OPERATION
     , UNDO_SQL
  FROM FLASHBACK_TRANSACTION_QUERY
 WHERE XID = HEXTORAW('02000D00E4070000');

OPERATION  UNDO_SQL
---------- ------------------------------------------------
DELETE     insert into "SCOTT"."MASTER"("PK","NAME") values
           ('1','andrew');
BEGIN
```

Note that there is always one row returned representing the start of the transaction. In this transaction, only one row was deleted in the master table. The `UNDO_SQL` column when executed will restore the original data into the table.

```
SQL> INSERT INTO "SCOTT"."MASTER"("PK","NAME") VALUES ('1','ANDREW');
SQL> COMMIT;
```

When you restart SQL Apply, the transaction will be applied to the standby database:

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

B

Upgrading Databases in a Data Guard Configuration

The procedures in this appendix describe how to upgrade to Oracle Database 11*g* Release 1 (11.1) when a physical or logical standby database is present in the configuration.

This appendix contains the following topics:

B.1 Before You Upgrade the Oracle Database Software

Consider the following points before beginning to upgrade your Oracle Database software:

- If you are using the Data Guard broker to manage your configuration, follow the instructions in the *Oracle Data Guard Broker* manual for information about removing or disabling the broker configuration.
- The procedures in this appendix are to be used in conjunction with the ones contained in the *Oracle Database Upgrade Guide* for 11*g* Release 1 (11.1).
- The procedures in this appendix use the Database Upgrade Assistant (DBUA) to perform the upgrade. For instructions on performing the upgrade manually, refer to the *Oracle Database Upgrade Guide*. The manual upgrade steps described should be performed whenever use of DBUA is mentioned.
- Check for nologging operations. If nologging operations have been performed then you must update the standby database. See Section 13.4, "Recovering After the NOLOGGING Clause Is Specified" for details.
- Make note of any tablespaces or datafiles that need recovery due to `OFFLINE IMMEDIATE`. Tablespaces or datafiles should be recovered and either online or offline prior to upgrading.

B.2 Upgrading Oracle Database with a Physical Standby Database In Place

Perform the following steps to upgrade to Oracle Database 11*g* Release 1 (11.1) when a physical standby database is present in the configuration:

1. Review and perform the steps listed in the "Preparing to Upgrade" chapter of the *Oracle Database Upgrade Guide*.
2. Shut down the primary database.
3. Shut down the physical standby database.
4. Install the new release of the Oracle software into a new Oracle home on the physical standby database system, as described in the *Oracle Database Upgrade Guide*.
5. Mount the physical standby database.

   Note: The standby database should not be opened until the primary database upgrade is completed.

6. Start Redo Apply on the physical standby database.
7. Install the new release of the Oracle software into a new Oracle home on the primary database system as described in the *Oracle Database Upgrade Guide*.
8. Upgrade the primary database as described in the *Oracle Database Upgrade Guide*. Note that the physical standby database will be upgraded when it applies the redo generated by the primary database as it is upgraded.
9. Open the upgraded primary database.
10. If Active Data Guard was being used prior to the upgrade, then refer to Section 9.2.1 for information about how to reenable it after upgrading.

B.3 Upgrading Oracle Database with a Logical Standby Database In Place

Note: This appendix describes the traditional method for upgrading your Oracle Database software with a logical standby database in place. A second method in Chapter 12, "Using SQL Apply to Upgrade the Oracle Database" describes how to upgrade with a logical standby database in place in a rolling fashion to minimize downtime. Use the steps from only one method to perform the complete upgrade. Do not attempt to use both methods or to combine the steps from the two methods as you perform the upgrade process.

The procedure described in this section assumes that the primary database is running in `MAXIMUM PERFORMANCE` data protection mode.

Perform the following steps to upgrade to Oracle Database 11*g* Release 1 (11.1) when a logical standby database is present in the configuration:

1. Review and perform the steps listed in the "Preparing to Upgrade" chapter of the *Oracle Database Upgrade Guide*.
2. Set the data protection mode to `MAXIMUM PERFORMANCE` at the primary database, if needed:

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

3. On the primary database, stop all user activity and defer the remote archival destination associated with the logical standby database (for this procedure, it is

assumed that LOG_ARCHIVE_DEST_2 is associated with the logical standby database):

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2=DEFER SCOPE=BOTH;
SQL> ALTER SYSTEM ARCHIVE LOG CURRENT;
```

4. Stop SQL Apply on the standby database:

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

5. On the primary database install the newer release of the Oracle software as described in the *Oracle Database Upgrade Guide*.
6. On the logical standby database, install the newer release of the Oracle software as described in *Oracle Database Upgrade Guide*.

> **Note:** Steps 5 and 6 can be performed concurrently (in other words, the primary and the standby databases can be upgraded concurrently) to reduce downtime during the upgrade procedure.

7. On the upgraded logical standby database, restart SQL Apply. If you are using Oracle RAC, start up the other standby database instances:

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

8. Open the upgraded primary database and allow users to connect. If you are using Oracle RAC, start up the other primary database instances.

   Also, enable archiving to the upgraded logical standby database, as follows:

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2=ENABLE;
```

9. Optionally, reset to the original data protection mode if you changed it in Step 2.

C

Data Type and DDL Support on a Logical Standby Database

When setting up a logical standby database, you must ensure the logical standby database can maintain the datatypes and tables in your primary database. This appendix lists the various database objects, storage types, and PL/SQL supplied packages that are supported and unsupported by logical standby databases. It contains the following topics:

C.1 Datatype Considerations

The following sections list the supported and unsupported database objects:

C.1.1 Supported Datatypes in a Logical Standby Database

Logical standby databases support the following datatypes:

`BINARY_DOUBLE`
`BINARY_FLOAT`
`BLOB`
`CHAR`
`CLOB` and `NCLOB`

DATE
INTERVAL YEAR TO MONTH
INTERVAL DAY TO SECOND
LONG
LONG RAW
NCHAR
NUMBER
NVARCHAR2
RAW
TIMESTAMP
TIMESTAMP WITH LOCAL TIMEZONE
TIMESTAMP WITH TIMEZONE
VARCHAR2 and VARCHAR
XMLType stored as CLOB

Note: SQL Apply support for the following datatypes has compatibility requirements on the primary database:

- Multibyte CLOB support (requires primary database to run at a compatibility of 10.1 or higher).
- IOT support without LOBs and Overflows (requires primary database to run at a compatibility of 10.1 or higher);
- IOT support with LOB and Overflow (requires primary database to run at a compatibility of 10.2 or higher)
- XMLType stored as CLOB (requires primary database to run at a compatibility of 11.1 or higher)
- TDE support (requires primary database to run at a compatibility of 11.1 or higher)

C.1.2 Unsupported Datatypes in a Logical Standby Database

Logical standby databases do not support the following datatypes:

BFILE
Collections (including VARRAYS and nested tables)
Multimedia data types (including Spatial, Image, and Oracle Text)
ROWID, UROWID
User-defined types
LOBs stored as SecureFiles
XMLType stored as Object Relational
Binary XML

C.2 Support for Transparent Data Encryption (TDE)

Data Guard SQL Apply can be used to provide data protection for a primary database with Transparent Data Encryption (TDE) enabled. Consider the following when using a logical standby database to provide data protection for applications with advanced security requirements:

- Tables with Transparent Data Encryption using server held keys are supported in the context of a logical standby database when both the primary and the standby databases are running at a compatibility level of 11.1 or higher.

- Transparent Data Encryption in the context of Hardware Security Modules is not supported in the context of a logical standby database in 11g Release 1.

You must consider the following restrictions when, in the context of a logical standby database, you want to replicate tables that have encrypted columns:

1. To translate encrypted redo records, SQL Apply must have access to an open wallet containing the Transparent Data Encryption keys. Therefore, you must copy the wallet containing the keys from the primary database to the standby database after it has been created.
2. The wallet must be copied from the primary database to the logical standby database every time the master key is changed.
3. Oracle recommends that you not rekey the master key at the logical standby database while the logical standby database is replicating encrypted tables from the primary database. Doing so may cause SQL Apply to halt when it encounters an encrypted redo record.
4. You can rekey the encryption key of a replicated table at the logical standby database. This requires that you lower the guard setting to `NONE` before you issue the rekey command.
5. Replicated encrypted tables can use a different encryption scheme for columns than the one used in the primary database. For example, if the `SALARY` column of the `HR.EMPLOYEES` table is encrypted at the primary database using the AES192 encryption algorithm, it can be encrypted at the logical standby using the AES256 encryption algorithm. Or, the `SALARY` column can remain unencrypted at the logical standby database.

C.3 Support for Tablespace Encryption

Data Guard SQL Apply can be used to provide data protection for a primary database that has tablespace encryption enabled. In such a case, restrictions 1, 2, and 3 listed in Section C.2, "Support for Transparent Data Encryption (TDE)" will apply.

Note: In some cases, when SQL Apply mines and applies redo records for changes made to tables in encrypted tablespaces, records of user data in unencrypted form may be kept for a long period of time. If this is not acceptable, you should issue the following command to move all metadata tables pertaining to the mining component of SQL Apply to an encrypted tablespace:

```
SQL> DBMS_LOGMNR_D.SET_TABLESPACE(NEW_TABLESPACE => 'ENCRYPTED_
LOGMNR_TS');
```

C.4 Support For Row-level Security and Fine-Grained Auditing

As of Oracle Database 11*g*, Logical Standby can automatically replicate the security environment provided through the `DBMS_RLS` and `DBMS_FGA` PL/SQL packages. This support simplifies management of security considerations when a server fails over to the standby since the security environment will transparently be maintained. It also ensures that access control policies applied to the primary data can be automatically forwarded to the standby, and the standby data transparently given the same level of protection. If a standby server is newly created with 11g, this replication is enabled by default; otherwise it has to be enabled by the DBA at an appropriate time.

Support for the replication of these PL/SQL packages requires that both the primary and the standby be running with a compatibility setting of 11.1 or higher.

It also requires that the table referenced be a Logical Standby maintained object (for example a table with a rowid column will not have its data maintained by Logical Standby, in which case DBMS_RLS and DBMS_FGA calls referencing that table will also not be maintained).

C.4.1 Row-level Security

Row-Level Security, also known as Virtual Private Database (VPD), is a security feature that enables you to enforce security to a fine level of granularity, directly on tables, views, or synonyms. When a user directly or indirectly accesses a table, view, or synonym protected with a VPD policy, the server dynamically modifies the SQL statement of the user. The modification creates a WHERE condition (known as a predicate) returned by a function implementing the security policy. The statement is modified dynamically, transparently to the user, using any condition that can be expressed in, or returned by, a function. VPD policies can be applied to SELECT, INSERT, UPDATE, INDEX, and DELETE statements. VPD is implemented by using the DBMS_RLS package to apply security policies.

When a DBMS_RLS procedure is executed on the primary, additional information is captured in the redo that allows the procedure call to be logically reconstructed and executed on the standby. Logical Standby supports replication of ancillary objects for VPD such as Contexts, Database Logon Triggers, and their supporting packages. You must ensure that these objects are placed in maintained schemas and that no DDL skips have been configured that would stop their replication.

C.4.2 Fine-Grained Auditing

Fine-grained auditing provides a way to audit select statements. The DBMS_FGA package enables all select statements that access a table to be captured, together with what data was accessed. An FGA policy may be applied to a particular column or even to only those select statements that return rows for which a specified predicate returns TRUE.

When a DBMS_FGA procedure is executed on the primary, additional information is captured to the redo that allows the procedure call to be logically reconstructed and executed on the standby.

C.4.3 Skipping and Enabling PL/SQL Replication

PL/SQL can be configured with skip and skip_error rules exactly as DDL statements except that wildcarding on the package and procedure are not supported. For example to skip all aspects of VPD, do the following:

```
DBMS_LOGSTDBY.Skip (
stmt => 'PL/SQL',
schema_name => 'SYS',
object_name =>'DBMS_RLS',
use_like => FALSE);
```

Note that the schema specified is the schema in which the package is defined. To skip an individual procedure in a package, the syntax would be as follows:

```
DBMS_LOGSTDBY.Skip (
stmt => 'PL/SQL',
schema_name => 'SYS',
object_name =>'DBMS_RLS.Add_Policy',
```

```
use_like => FALSE);
```

In order to skip VPD on certain schemas or tables, a skip procedure must be used. The skip procedure will be passed the fully qualified PL/SQL statement that is to be executed, for example:

```
DBMS_RLS.Drop_Policy(
object_schema => 'SCOTT,
object_name  => 'EMP',
policy_name => 'MYPOLICY');
```

The procedure could then parse the statement to decide whether to skip it, to apply it, or to stop apply and let the DBA take a compensating action.

Unlike DDL, skip procedures on PL/SQL do not support returning a replacement statement.

C.5 Oracle Label Security

Logical standby databases do not support Oracle Label Security. If Oracle Label Security is installed on the primary database, SQL Apply fails on the logical standby database with an internal error during startup.

C.6 Supported Table Storage Types

Logical standby databases support the following table storage types:

- Cluster tables (including index clusters and heap clusters)
- Index-organized tables (partitioned and nonpartitioned, including overflow segments)
- Heap-organized tables (partitioned and nonpartitioned)

C.7 Unsupported Table Storage Types

Logical standby databases do not support the following table storage types:

- Tables stored with segment compression enabled
- Tables containing LOB columns stored as SecureFiles
- Tables with virtual columns

C.8 PL/SQL Supplied Packages Considerations

This section discusses the following considerations regarding PL/SQL supplied packages:

- Supported PL/SQL Supplied Packages
- Unsupported PL/SQL Supplied Packages
- Handling XML and XDB PL/SQL Packages in Logical Standby

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

C.8.1 Supported PL/SQL Supplied Packages

Oracle PL/SQL supplied packages that do not modify system metadata or user data leave no footprint in the archived redo log files, and hence are safe to use on the primary database. Examples of such packages are DBMS_OUTPUT, DBMS_RANDOM, DBMS_PIPE, DBMS_DESCRIBE, DBMS_OBFUSCATION_TOOLKIT, DBMS_TRACE, DBMS_METADATA, DBMS_CRYPTO.

Oracle PL/SQL supplied packages that do not modify system metadata but may modify user data are supported by SQL Apply, as long as the modified data belongs to the supported data types listed in Section C.1.1. Examples of such packages are DBMS_LOB, DBMS_SQL, and DBMS_TRANSACTION.

Data Guard logical standby supports replication of actions performed through the following two packages: DBMS_RLS and DBMS_FGA.

C.8.2 Unsupported PL/SQL Supplied Packages

Oracle PL/SQL supplied packages that modify system metadata typically are not supported by SQL Apply, and therefore their effects are not visible on the logical standby database. Examples of such packages are DBMS_JAVA, DBMS_REGISTRY, DBMS_ALERT, DBMS_SPACE_ADMIN, DBMS_REFRESH, DBMS_REDEFINITION, and DBMS_AQ.

Specific support for DBMS_JOB has been provided. Job execution is suspended on a logical standby database and jobs cannot be scheduled directly on the standby database. However, jobs submitted on the primary database are replicated in the standby database. In the event of a switchover or failover, jobs scheduled on the original primary database will automatically begin running on the new primary database.

Specific support for DBMS_SCHEDULER has been provided to allow jobs to be run on a standby database. A new attribute of a scheduler job has been created in 11*g* called database_role whose contents match the database_role attribute of V$DATABASE. When a scheduler job is created, it defaults to the local role (that is, a job created on the standby defaults to a database_role of LOGICAL STANDBY). The job scheduler executes only jobs specific to the current role. On switchover or failover, the scheduler automatically switches to running jobs specific to the new role.

Scheduler jobs are not replicated to the standby. However, existing jobs can be activated under the new role by using the DBMS_SCHEDULER.Set_Attribute procedure. Alternatively, jobs that should run in both roles can be cloned and the copy made specific to the other role. The DBA_SCHEDULER_JOB_ROLES view shows which jobs are specific to which role.

Scheduler jobs obey the database guard when they run on a logical standby database. Thus, in order to run jobs that need to modify unmaintained tables, the database guard should be set to STANDBY. (It is not possible to use the ALTER SESSION DISABLE GUARD statement inside a PL/SQL block and have it take effect.)

> **See Also:** *Oracle Database PL/SQL Packages and Types Reference* for details about specific packages

C.8.3 Handling XML and XDB PL/SQL Packages in Logical Standby

In Oracle Database 11*g* release 1 (11.1), Logical Standby supports XML when it is stored in CLOB format. However, there are several PL/SQL packages used in conjunction with XML that are not fully supported.

The PL/SQL packages and procedures that are supported by Logical Standby only modify in-memory structures; they do not modify data stored in the database. These packages do not generate redo and therefore are not replicated to a Logical Standby.

Certain PL/SQL packages and procedures related to XML and XDB that are not supported by Logical Standby, but that require corresponding invocations at the logical standby database for replication activities to continue, are instrumented such that invocations of these procedures at the primary database will generate additional redo records indicating procedure invocation. When SQL Apply encounters such redo records, it stops and writes an error message in the `DBA_LOGSTDBY_EVENTS` table, indicating the procedure name. This allows the DBA to invoke the corresponding procedure at the logical standby database at the appropriate time so that subsequent redo records generated at the primary database can be applied successfully at the logical standby database. See Section C.8.3.1 through Section C.8.3.6 for more information about dealing with these unsupported procedures.

The following packages contain unsupported procedures:

- `DBMS_XMLSCHEMA`
- `DBMS_XMLINDEX`

In addition to these packages, Logical Standby does not support any modifications to the XDB schema. The objects within the XDB schema are considered to be system metadata and direct modifications to them are not replicated.

Tables managed by the Oracle XML DB Repository, also known as hierarchy-enabled tables, are not supported by Logical Standby. These tables are used to store XML data and can be accessed using the FTP and HTTP protocols, as well as the normal SQL access. For more information on these tables, refer to the *Oracle XML DB Developer's Guide*.

C.8.3.1 The DBMS_XMLSCHEMA Schema

The following procedures within the DBMS_XMLSCHEMA package are unsupported and cannot be replicated by Logical Standby. Logical Standby stops when it encounters calls to these procedures to provide the user an opportunity to take a compensating action for these calls. Sections Section C.8.3.3 through Section C.8.3.6 provide more information on the alternatives available for dealing with these unsupported procedures.

- `REGISTERSCHEMA`
- `REGISTERURI`
- `DELETESCHEMA`
- `PURGESCHEMA`
- `COPYEVOLVE`
- `INPLACEEVOLVE`
- `COMPILESCHEMA`

The XDB schema is an Oracle managed schema. Any changes to this schema are automatically skipped by Logical Standby. The following procedure makes changes to the XDB schema which will not be replicated:

- `GENERATEBEAN`

The following procedures and functions do not generate redo and therefore do not stop Logical Standby:

- GENERATESCHEMAS
- GENERATESCHEMA

C.8.3.2 The DBMS_XMLINDEX Package

The SYNCINDEX procedure within the DBMS_XMLINDEX package is marked as unsupported and cannot be replicated by Logical Standby. Logical Standby stops when it encounters calls to it.

The following functions and procedures do not generate redo and therefore do not stop Logical Standby:

- NODEREFGETREF
- NODEREFGETVALUE
- NODEREFGETPARENTREF
- NODEREFGETNAME
- NODEREFGETNAMESPACE

C.8.3.3 Dealing With Unsupported PL/SQL Procedures

There are a couple options for dealing with unsupported PL/SQL procedures. The first option is to allow the Logical Standby apply process to stop and to manually perform some compensating action. The second option is to take a preemptive action and to skip the unsupported PL/SQL either by using Logical Standby skip procedures. Each of these options is discussed in the following sections.

C.8.3.4 Manually Compensating for Unsupported PL/SQL

When Logical Standby encounters something that is unsupported, it stops the apply process and records an error in the DBA_LOGSTDBY_EVENTS table. You can query this table to determine what action caused the standby to stop and what action, if any, needs to be taken to compensate.

The following example shows a sample of what this query and its output might look like:

```
select status, event from dba_logstdby_events
          where commit_scn >= (select applied_scn from dba_logstdby_progress) and
          status_code = 16265
          order by commit_scn desc;

STATUS
--------------------------------------------------------------------------------
EVENT
--------------------------------------------------------------------------------
ORA-16265: Unsupported PL/SQL procedure encountered
begin
 "XDB"."DBMS_XMLSCHEMA"."REGISTERSCHEMA" (
   "SCHEMAURL" => 'xmlplsqlsch2

ORA-16265: Unsupported PL/SQL procedure encountered
begin
 "XDB"."DBMS_XMLSCHEMA"."REGISTERSCHEMA" (
   "SCHEMAURL" => 'xmlplsqlsch2


2 rows selected.
```

Two rows with the same information are returned because Logical Standby automatically retries the failed transaction. The results show that the standby was stopped when a call to DBMS_XMLSCHEMA.REGISTERSCHEMA was encountered for the xmlplsqlsch2 schema. You can use this information to transfer any needed files from the primary and register the schema on the standby.

Once the schema has been successfully registered on the standby, the apply process on the Logical Standby can be restarted. This must be performed using the SKIP FAILED TRANSACTION option, for example:

```
alter database start logical standby apply skip failed transaction'
```

Logical Standby skips past the offending transaction and continues applying redo from the primary.

The general procedure for manually replicating unsupported PL/SQL follows these steps:

1. Some unsupported PL/SQL is executed on the primary database.
2. The standby database encounters the unsupported PL/SQL and stops Apply.
3. You examine the DBA_LOGSTDBY_EVENTS table to determine what caused Apply to stop.
4. You execute some compensating actions on the standby for the unsupported PL/SQL.
5. You restart apply on the standby.

C.8.3.5 Proactively Compensating for Unsupported PL/SQL

In certain cases, you know that an action you are going to perform on the primary database will cause the standby to halt. In those cases, you may want to take action ahead of time to either minimize or eliminate the time that the standby is not applying redo.

For example, suppose you know that a new application is going to be installed. Part of the installation requires a large number of XML schemas to be registered. You can register these schemas on the standby before they are registered on the primary. You can also install a skip procedure on the standby for the DBMS_XMLSCHEMA.REGISTERSCHEMA procedure which will check to see if the XML schema is registered and if so, it will tell Logical Standby to skip that PL/SQL call.

This approach can also be used for some of the other PL/SQL procedures that are unsupported. For example, DBMS_XMLSCHEMA.DELETESCHEMA can be handled in a similar way. A skip procedure can be written to see if the schema is installed on the standby and if it is not, then that PL/SQL can be safely skipped because it would not have had any meaningful affect on the standby.

C.8.3.6 Compensating for Ordering Sensitive Unsupported PL/SQL

Although the previous approach is useful, it cannot be used in all cases. It can only be safely used when the time that the PL/SQL is executed relative to other transactions is not critical. One case that this should not be used for is that of DBMS_XMLSCHEMA.copyEvolve.

This procedure evolves, or changes, a schema and can modify tables by adding and or removing columns and it can also change whether or not XML documents are valid. The timing of when this procedure should be executed on the Logical Standby is critical. The only time guaranteed to be safe is when apply has stopped on the Logical Standby when it sees that this procedure was executed on the primary database.

Before evolving a schema, it is also important to quiesce any traffic on the primary that may be using the schema. Otherwise, a transaction that is executed close in time to the evolveSchema on the primary may be executed in a different order on the Logical Standby because the dependency between the two transactions is not apparent to the Logical Standby. Therefore, when ordering sensitive PL/SQL is involved, you should follow these steps:

1. Quiesce changes to dependent tables on the primary.
2. Execute the CopyEvolve on the primary.
3. Wait for the standby to stop on the CopyEvolve PL/SQL.
4. Apply the compensating CopyEvolve on the standby.
5. Restart apply on the standby.

Example C–1 shows a sample of the procedures that could be used to determine how to handle RegisterSchema calls.

Example C–1 PL/SQL Skip Procedure for RegisterSchema

```
-- Procedures to determine how to handle registerSchema calls

-- This procedure extracts the schema URL, or name, from the statement
-- string that is passed into the skip procedure.

Create or replace procedure sec_mgr.parse_schema_str(
  statement             in varchar2,
  schema_name      out varchar2)
Is
  pos1 number;
  pos2 number;
  workingstr   varchar2(32767);
Begin

-- Find the correct argument
pos1 := instr(statement, '"SCHEMAURL" => ''');
workingstr := substr(statement, pos1 + 16);

-- Find the end of the schema name
pos1 := instr(workingstr, '''');

-- Get just the schema name
workingstr := substr(workingstr, 1, pos1 - 1);

schema_name := workingstr;

End parse_schema_str;
/
show errors


-- This procedure checks if a schema is already registered. If so,
-- it returns the value DBMS_LOGSTDBY.SKIP_ACTION_SKIP to indicate that
-- the PL/SQL should be skipped. Otherwise, the value
-- DBMS_LOGSTDBY.SKIP_ACTION_SKIP is returned and Logical Standby apply
-- will halt to allow the DBA to deal with the registerSchema call.

Create or replace procedure sec_mgr.skip_registerschema(
  statement          in varchar2,
  package_owner            in varchar2,
```

```
  package_name             in varchar2,
  procedure_name          in varchar2,
  current_user              in varchar2,
  xidusn             in number,
  xidslt               in number,
  xidsqn             in number,
  exit_status           in number,
  skip_action      out number)
Is
  schema_exists number;
  schemastr varchar2(2000);
Begin

  skip_action := DBMS_LOGSTDBY.SKIP_ACTION_SKIP;

  -- get the schame name from statement
  parse_schema_str(statement, schemastr);

  -- see if the schema is already registered
  select count(*) into schema_exists from sys.all_xml_schemas s
                                     where s.schema_url = schemastr and
                                           s.owner = current_user;

  IF schema_exists = 0 THEN
      -- if the schema is not  registered, then we must stop apply
      skip_action := DBMS_LOGSTDBY.SKIP_ACTION_APPLY;
  ELSE
      -- if the schema is already registered, then we can skip this statement
      skip_action := DBMS_LOGSTDBY.SKIP_ACTION_SKIP;
  END IF;

End skip_registerschema;
/
show errors

-- Register the skip procedure to deal with the unsupported registerSchema
-- PL/SQL.
Begin
   sys.dbms_logstdby.skip(stmt => 'PL/SQL',
        schema_name => 'XDB',
                object_name   => 'DBMS_XMLSCHEMA.REGISTERSCHEMA',
                          proc_name     => 'SEC_MGR.SKIP_REGISTERSCHEMA',
        use_like         => FALSE );
                  End;
    /
show errors
```

C.9 Unsupported Tables

It is important to identify unsupported database objects on the primary database before you create a logical standby database because changes made to unsupported data types and tables on the primary database will be automatically skipped by SQL Apply on the logical standby database. Moreover, no error message will be returned.

There are three types of objects on a database, from the perspective of logical standby support:

- Objects that are explicitly maintained by SQL Apply
- Objects that are implicitly maintained by SQL Apply

- Objects that are not maintained by SQL Apply

Some schemas that ship with the Oracle database (for example, SYSTEM) contain objects that will be implicitly maintained by SQL Apply. However, if you put a user-defined table in SYSTEM, it will not be maintained even if it has columns of supported data types. To discover which objects are not maintained by SQL Apply, you must run two queries. The first query is as follows:

```
SQL> SELECT OWNER FROM DBA_LOGSTDBY_SKIP WHERE STATEMENT_OPT = 'INTERNAL SCHEMA';
```

This will return all schemas that are considered to be internal. User tables placed in these schemas will not be replicated on a logical standby database and will not show up in the DBA_LOGSTDBY_UNSUPPORTED view. Tables in these schemas that are created by Oracle will be maintained on a logical standby, if the feature implemented in the schema is supported in the context of logical standby.

The second query you must run is as follows. It returns tables that do not belong to internal schemas and will not be maintained by SQL Apply because of unsupported data types:

```
SQL> SELECT DISTINCT OWNER,TABLE_NAME FROM DBA_LOGSTDBY_UNSUPPORTED
  2> ORDER BY OWNER,TABLE_NAME;

OWNER        TABLE_NAME
-----------  --------------------------
HR           COUNTRIES
OE           ORDERS
OE           CUSTOMERS
OE           WAREHOUSES
```

To view the column names and data types for one of the tables listed in the previous query, use a SELECT statement similar to the following:

```
SQL> SELECT COLUMN_NAME,DATA_TYPE FROM DBA_LOGSTDBY_UNSUPPORTED
  2> WHERE OWNER='OE' AND TABLE_NAME = 'CUSTOMERS';

COLUMN_NAME                    DATA_TYPE
-----------------------------  -------------------
CUST_ADDRESS                   CUST_ADDRESS_TYP
PHONE_NUMBERS                  PHONE_LIST_TYP
CUST_GEO_LOCATION              SDO_GEOMETRY
```

If the primary database contains unsupported tables, SQL Apply automatically excludes these tables when applying redo data to the logical standby database.

> **Note:** If you determine that the critical tables in your primary database will not be supported on a logical standby database, then you might want to consider using a physical standby database. Physical standby databases do not have any such data type restrictions.

C.10 Skipped SQL Statements on a Logical Standby Database

By default, the following SQL statements are automatically skipped by SQL Apply:

```
ALTER DATABASE
ALTER MATERIALIZED VIEW
ALTER MATERIALIZED VIEW LOG
ALTER SESSION
```

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

All other SQL statements executed on the primary database are applied to the logical standby database.

C.11 DDL Statements Supported by a Logical Standby Database

Table C–1 lists the supported values for the `stmt` parameter of the `DBMS_LOGSTDBY.SKIP` procedure. The left column of the table lists the keywords that may be used to identify the set of SQL statements to the right of the keyword. In addition, any of the SQL statements listed in the `sys.audit_actions` table (shown in the right column of Table 1-13) are also valid values. Note that keywords are generally defined by database object.

> **See Also:** *Oracle Database PL/SQL Packages and Types Reference* for complete information about the `DBMS_LOGSTDBY` package and Section 10.5.3, "Setting up a Skip Handler for a DDL Statement"

Table C–1 Values for stmt Parameter of the DBMS_LOGSTDBY.SKIP procedure

Keyword	Associated SQL Statements
There is no keyword for this group of SQL statements.	GRANT OBJECT REVOKE OBJECT SYSTEM GRANT SYSTEM REVOKE
CLUSTER	AUDIT CLUSTER CREATE CLUSTER DROP CLUSTER TRUNCATE CLUSTER
CONTEXT	CREATE CONTEXT DROP CONTEXT
DATABASE LINK	CREATE DATABASE LINK CREATE PUBLIC DATABASE LINK DROP DATABASE LINK DROP PUBLIC DATABASE LINK
DIMENSION	ALTER DIMENSION CREATE DIMENSION DROP DIMENSION

Table C–1 (Cont.) Values for stmt Parameter of the DBMS_LOGSTDBY.SKIP procedure

Keyword	Associated SQL Statements
DIRECTORY	CREATE DIRECTORY DROP DIRECTORY
DML	Includes DML statements on a table (for example: INSERT, UPDATE, and DELETE)
INDEX	ALTER INDEX CREATE INDEX DROP INDEX
NON_SCHEMA_DDL	*All DDL that does not pertain to a particular schema* **Note:** SCHEMA_NAME and OBJECT_NAME must be null
PROCEDURE[1]	ALTER FUNCTION ALTER PACKAGE ALTER PACKAGE BODY ALTER PROCEDURE CREATE FUNCTION CREATE LIBRARY CREATE PACKAGE CREATE PACKAGE BODY CREATE PROCEDURE DROP FUNCTION DROP LIBRARY DROP PACKAGE DROP PACKAGE BODY DROP PROCEDURE
PROFILE	ALTER PROFILE CREATE PROFILE DROP PROFILE
PUBLIC DATABASE LINK	CREATE PUBLIC DATABASE LINK DROP PUBLIC DATABASE LINK
PUBLIC SYNONYM	CREATE PUBLIC SYNONYM DROP PUBLIC SYNONYM
ROLE	ALTER ROLE CREATE ROLE DROP ROLE SET ROLE
ROLLBACK STATEMENT	ALTER ROLLBACK SEGMENT CREATE ROLLBACK SEGMENT DROP ROLLBACK SEGMENT
SCHEMA_DDL	*All DDL statements that create, modify, or drop schema objects (for example: tables, indexes, and columns)* **Note:** SCHEMA_NAME and OBJECT_NAME must *not* be null
SEQUENCE	ALTER SEQUENCE CREATE SEQUENCE DROP SEQUENCE
SYNONYM	CREATE PUBLIC SYNONYM CREATE SYNONYM DROP PUBLIC SYNONYM DROP SYNONYM
SYSTEM AUDIT	AUDIT *SQL_statements* NOAUDIT *SQL_statements*

Table C–1 (Cont.) Values for stmt Parameter of the DBMS_LOGSTDBY.SKIP procedure

Keyword	Associated SQL Statements
TABLE	CREATE TABLE DROP TABLE TRUNCATE TABLE
TABLESPACE	CREATE TABLESPACE DROP TABLESPACE ALTER TABLESPACE
TRIGGER	ALTER TRIGGER CREATE TRIGGER DISABLE ALL TRIGGERS DISABLE TRIGGER DROP TRIGGER ENABLE ALL TRIGGERS ENABLE TRIGGER
TYPE	ALTER TYPE ALTER TYPE BODY CREATE TYPE CREATE TYPE BODY DROP TYPE DROP TYPE BODY
USER	ALTER USER CREATE USER DROP USER
VIEW	CREATE VIEW DROP VIEW

[1] Java schema objects (sources, classes, and resources) are considered the same as procedures for purposes of skipping (ignoring) SQL statements.

See Also: The following sections that provide usage examples of the `SKIP` and `UNSKIP` options:

- Section 10.5.2, "Using DBMS_LOGSTDBY.SKIP to Prevent Changes to Specific Schema Objects"
- Section 10.5.3, "Setting up a Skip Handler for a DDL Statement"
- Section 10.5.4, "Modifying a Logical Standby Database"
- Section 10.5.5, "Adding or Re-Creating Tables On a Logical Standby Database"

C.11.1 DDL Statements that Use DBLINKS

SQL Apply may not correctly apply DDL statements such as the following, that reference a database link:

```
CREATE TABLE tablename AS SELECT * FROM bar@dblink
```

This is because the *dblink* at the logical standby database may not point to the same database as the primary database. If SQL Apply fails while executing such a DDL statement, you should use the `DBMS_LOGSTDBY.INSTANTIATE_TABLE` procedure for the table being created, and then restart SQL APPLY operations.

C.11.2 Replication of AUD$ and FGA_LOG$ on Logical Standbys

Auditing and fine-grained auditing are supported on logical standbys. Changes made to the AUD$ and FGA_AUD$ tables at the primary database are replicated at the logical standby.

Both the AUD$ table and the FGA_AUD$ table have a DBID column. If the DBID value is that of the primary database, then the row was replicated to the logical standby based on activities at the primary. If the DBID value is that of the logical standby database, then the row was inserted as a result of local activities at the logical standby.

After the logical standby database assumes the primary role as a result of a role transition (either a switchover or failover), the AUD$ and FGA_AUD$ tables at the *new primary* (originally the logical standby) and at the *new logical standby* (originally the primary) are not necessarily synchronized. Therefore, it is possible that not all rows in the AUD$ or FGA_AUD$ tables at the new primary database will be present in the new logical standby database. However, all rows in AUD$ and FGA_LOG$ that were inserted while the database was in a primary role are replicated and present in the logical standby database.

D

Data Guard and Oracle Real Application Clusters

An Oracle Data Guard configuration can consist of any combination of single-instance and Oracle Real Application Clusters (RAC) multiple-instance databases. This chapter summarizes the configuration requirements and considerations that apply when using Oracle Data Guard with Oracle RAC databases. It contains the following sections:

- Configuring Standby Databases in an Oracle RAC Environment
- Configuration Considerations in an Oracle RAC Environment
- Troubleshooting

D.1 Configuring Standby Databases in an Oracle RAC Environment

You can configure a standby database to protect a primary database using Oracle RAC. The following table describes the possible combinations of instances in the primary and standby databases:

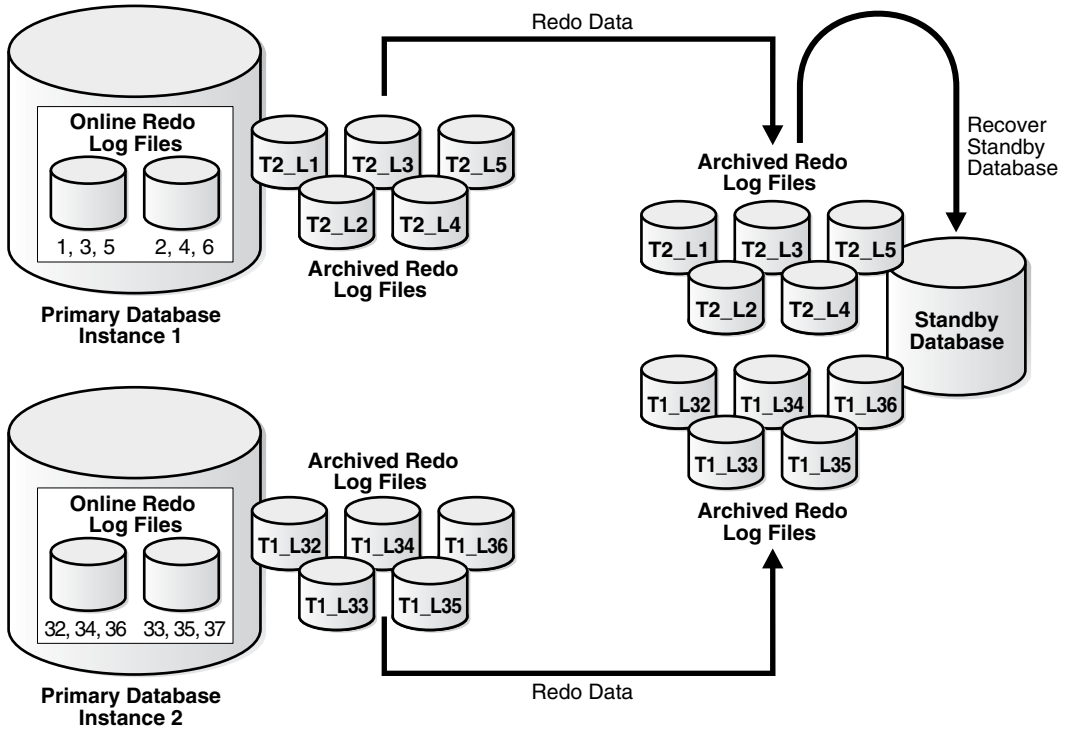
Instance Combinations	Single-Instance Standby Database	Multi-Instance Standby Database
Single-instance primary database	Yes	Yes
Multi-instance primary database	Yes	Yes

In each scenario, each instance of the primary database transmits its redo data to an instance of the standby database.

D.1.1 Setting Up a Multi-Instance Primary with a Single-Instance Standby

Figure D–1 illustrates an Oracle RAC database with two primary database instances (a multi-instance primary database) transmitting redo data to a single-instance standby database.

Figure D–1 Transmitting Redo Data from a Multi-Instance Primary Database



In this case, Instance 1 of the primary database archives redo data to local archived redo log files 1, 2, 3, 4, 5 and transmits the redo data to the standby database destination, while Instance 2 archives redo data to local archived redo log files 32, 33, 34, 35, 36 and transmits the redo data to the same standby database destination. The standby database automatically determines the correct order in which to apply the archived redo log files.

To set up a primary database in an Oracle RAC 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 `LOG_ARCHIVE_DEST_`*n* and `LOG_ARCHIVE_FORMAT` parameters to specify the location of the archived redo log files and standby redo log files.

D.1.2 Setting Up Oracle RAC Primary and Standby Databases

This section describes how to configure an Oracle RAC primary database to send redo data to an Oracle RAC standby database.

D.1.2.1 Configuring an Oracle RAC Standby Database to Receive Redo Data

Perform the following steps to configure an Oracle RAC standby database to receive redo data from a primary database:

1. Create a standby redo log on the standby database. The redo log files in the standby redo log must reside in a location that can be accessed by all of the

standby database instances, such as on a cluster file system or ASM instance. See Section 6.2.3.1 for more information about creating a standby redo log.

2. Configure standby redo log archival on each standby database instance. The standby redo log must be archived to a location that can be accessed by all of the standby database instances, and every standby database instance must be configured to archive the standby redo log to the same location. See Section 6.2.3.2 for more information about configuring standby redo log archival.

D.1.2.2 Configuring an Oracle RAC Primary Database to Send Redo Data

Configure each instance of the RAC primary database to send its redo data to the RAC standby database. Section 6.2.2 describes how to configure an Oracle database instance to send redo data to another database.

Oracle recommends the following best practices when configuring an Oracle RAC primary database to send redo data to an Oracle RAC standby database:

1. Use the same `LOG_ARCHIVE_DEST_n` parameter on each primary database instance to send redo data to a given standby database.
2. Set the `SERVICE` attribute of each `LOG_ARCHIVE_DEST_n` parameter that corresponds to a given standby database to the same net service name.
3. The net service name should resolve to an Oracle Net connect descriptor that contains an address list, and that address list should contain connection data for each standby database instance.

D.2 Configuration Considerations in an Oracle RAC Environment

This section contains the Data Guard configuration information that is specific to Oracle RAC environments. It contains the following topics:

- Format for Archived Redo Log Filenames
- Data Protection Modes
- Role Transitions

D.2.1 Format for Archived Redo Log Filenames

The format for archived redo log filenames is in the form of log_*%parameter*, where *%parameter* can include one or more of the parameters in Table D–1.

Table D–1 Directives for the LOG_ARCHIVE_FORMAT Initialization Parameter

Directives	Description
`%a`	Database activation ID.
`%A`	Database activation ID, zero filled.
`%d`	Database ID.
`%D`	Database ID, zero filled.
`%t`	Instance thread number.
`%T`	Instance thread number, zero filled.
`%s`	Log file sequence number.
`%S`	Log file sequence number, zero filled.
`%r`	Resetlogs ID.

Table D–1 (Cont.) Directives for the LOG_ARCHIVE_FORMAT Initialization Parameter

Directives	Description
%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 Oracle RAC to uniquely identify the archived redo log files with the LOG_ARCHIVE_FORMAT parameter.

D.2.2 Data Protection Modes

In an Oracle RAC configuration when running in either maximum protection or maximum availability mode, any instance that loses connectivity with a standby destination will cause all other instances to stop sending data to that destination (this maintains the integrity of the data that has been transmitted to that destination).

When the failed standby destination comes back up, Data Guard runs the site in resynchronization mode until no gaps remain. Then, the standby destination can participate in the Data Guard configuration again.

The following list describes the behavior of the protection modes in Oracle RAC environments:

- Maximum protection configuration

  If a lost destination is the *last* participating SYNC destination, the instance loses connectivity and will be shut down. Other instances in an Oracle RAC 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 an Oracle RAC configuration loses connectivity to the last standby destination will the primary database be shut down.

D.2.3 Role Transitions

This section contains the following topics:

- Switchovers
- Failovers

D.2.3.1 Switchovers

For an Oracle RAC database, only one primary instance and one standby instance can be active during a switchover where the target database is a physical standby. Therefore, before a switchover to a physical standby database, 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. This limitation does not exist for a logical standby database.

Note: The SQL ALTER DATABASE statement used to perform the switchover automatically creates redo log files if they do not already exist. Because this can significantly increase the time required to complete the COMMIT operation, Oracle recommends that you manually add redo log files when creating physical standby databases.

D.2.3.2 Failovers

Before performing a failover to an Oracle RAC standby database, first shut down all but one standby instance. After the failover completes, restart the instances that were shut down.

D.3 Troubleshooting

This section provides help troubleshooting problems with Oracle RAC.

D.3.1 Switchover Fails in an Oracle RAC Configuration

When your database is using Oracle RAC, 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@standby2 AS SYSDBA
Enter Password:
SQL> SHUTDOWN;
SQL> EXIT
```

E

Cascaded Destinations

To reduce the load on your primary system, or to reduce the bandwidth requirements imposed when your standbys are separated from the primary database through a Wide Area Network (WAN), you can implement cascaded destinations, whereby a standby database receives its redo data from another standby database, instead of directly from the primary database.

In a Data Guard configuration using a cascaded destination, a physical standby database can forward the redo data it receives from the primary database to another standby database. Only a physical standby database can be configured to forward redo data to another standby database. A logical standby database cannot forward redo to another standby database.

> **Note:** You cannot set up a physical standby to forward redo if the primary database is part of an Oracle Real Application Clusters (RAC) environment or a Data Guard Broker environment.

The following Data Guard configurations using cascaded destinations are supported:

- Primary Database > Physical Standby Database with cascaded destination > Physical Standby Database
- Primary Database > Physical Standby Database with cascaded destination > Logical Standby Database

A physical standby database can support a maximum of nine remote destinations. When a cascaded destination is defined on a physical standby database, the physical standby will forward redo it receives from the primary to a second standby database after its standby redo log becomes full and is archived. Thus, the second standby database receiving the forwarded redo as a result of a cascaded destination will necessarily lag behind the primary database. Oracle recommends that cascaded destinations be used only for offloading reporting or for applications that do not require access to data that is completely up-to-date with the primary system. This is because the very nature of a cascaded destination means that the standby database that is the end-point will be one or more log files behind the primary database. Oracle also recommends that standby databases whose primary role is to be involved in role transitions receive their redo data directly from the primary database.

The rest of this appendix contains information about the following:

E.1 Configuring Cascaded Destinations

To enable a physical standby database to forward incoming redo data to a cascaded destination, perform the following steps:

1. Create standby redo log files on the physical standby database (if not already created).
2. If standby redo log files are not already defined, you can define them dynamically on the standby database. The standby database will begin using them after the next log switch on the primary database.
3. Define a LOG_ARCHIVE_DEST_*n* initialization parameter on the primary database to set up a physical standby database that will forward redo to a cascaded destination. Define the destination to use:
   - ASYNC or SYNC
   - Optionally, set the VALID_FOR attribute so that redo forwarding is enabled even after a role transition happens between the original primary database and the intermediate standby database that is forwarding redo. This may be meaningful in cases where the databases are separated over Wide Area Networks.
4. Ensure that archiving is enabled on the physical standby database where the cascaded destinations are defined (the standby database that will forward redo).
5. Configure a LOG_ARCHIVE_DEST_*n* parameter (on the physical standby that will forward redo data) for each cascaded destination.

Example E–1 shows the initialization parameters for a primary database named Boston, which sends redo to a physical standby database named Chicago, that forwards the redo it receives to a cascaded standby database named Denver. In this example, the database named Denver is a logical standby database, but note that a physical standby database can forward redo to either a physical or a logical standby database.

> **Note:** When the cascaded destination is a logical standby database, remember that you will create it just as if the logical standby will be directly connected to the primary database. See Chapter 4, "Creating a Logical Standby Database" for more information.

Example E–1 Sample Use of Initialization Parameters in Cascaded Destinations

Boston Database (Primary Role)

```
DB_UNIQUE_NAME=boston
REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE

LOG_ARCHIVE_CONFIG='DG_CONFIG=(chicago,boston,denver)'

LOG_ARCHIVE_DEST_1='LOCATION=/arch1/boston/ VALID_FOR=(ALL_LOGFILES,PRIMARY_ROLE)
DB_UNIQUE_NAME=boston'

LOG_ARCHIVE_DEST_2= 'SERVICE=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'
```

Chicago Database (Standby Role)

```
DB_UNIQUE_NAME=chicago
LOG_ARCHIVE_CONFIG= 'DG_CONFIG=(chicago,boston,denver)'
REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE

LOG_ARCHIVE_DEST_1= 'LOCATION=/arch1/chicago/  VALID_FOR=(ALL_LOGFILES,ALL_ROLES)
DB_UNIQUE_NAME=chicago'

LOG_ARCHIVE_DEST_2= 'SERVICE=denver  VALID_FOR=(STANDBY_LOGFILES,STANDBY_ROLE)
DB_UNIQUE_NAME=denver'

LOG_ARCHIVE_DEST_3=  'SERVICE=boston  VALID_FOR=  (ONLINE_LOGFILES,PRIMARY_ROLE)
DB_UNIQUE_NAME=boston'
```

Denver Database (Standby Role)

```
DB_UNIQUE_NAME=denver
LOG_ARCHIVE_CONFIG= 'DG_CONFIG=(chicago,boston,denver)'
REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE

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'
```

Both the Boston primary database and the Chicago physical standby database define the `LOG_ARCHIVE_DEST_2` initialization parameter as `SERVICE=denver VALID_FOR=(STANDBY_LOGFILES, STANDBY_ROLE)`. Hence, even if the Boston and Chicago databases switch roles, the redo data will continue to be forwarded to the Denver database. Remember, as part of the original setup of the physical standby database, you should define a local destination, `VALID_FOR=(ALL_LOGFILES, PRIMARY_ROLE)`, that will be used for local archiving when the physical standby database transitions to the primary role.

E.2 Role Transitions with Cascaded Destinations

Oracle recommends that standby databases primarily intended for disaster recovery purposes receive redo data directly from the primary database. This will result in the optimum level of data protection. A cascaded destination may be used as a second line of defense, but by definition it will always be further behind than a standby database that is receiving redo directly from the primary.

E.3 Examples of Using Cascaded Destinations

This section describes the following scenarios which demonstrate configuration options and uses for cascaded destinations:

- Physical Standby Forwarding Redo to a Remote Physical Standby
- Physical Standby Forwarding Redo to a Logical Standby

E.3.1 Physical Standby Forwarding Redo to a Remote Physical Standby

You have a primary database in your corporate offices, and you want to create a standby database at another facility within your metropolitan area to provide zero data loss protection if there is a failure at your primary site. In addition to the local standby, you wish to maintain a geographically remote standby database 2000 miles

away at a disaster recovery site. A small amount of data loss is acceptable if failover to the remote standby is required (an acceptable trade-off in return for the extra protection against events that can affect a large geographic area and cause both the primary site and the local standby database to fail). The remote standby database also provides continuous data protection after a failover to the local standby database and improves security by enabling backups to be created and stored at the remote location, eliminating the need to ship tapes off-site.

You could configure your primary database to ship redo directly to both standby databases; however, you may want to eliminate the potential overhead of the primary database shipping redo over a WAN to the second standby database. You solve this problem by creating the first physical standby in a local facility within your metropolitan area using the SYNC network transport to achieve zero data loss protection. A cascaded destination is defined on the local physical standby database that will forward redo received from the primary to the remote standby database using ASYNC network transport. Because the local standby manages all communication with the remote standby via a cascaded destination, there is no impact on the primary database to maintain a second standby.

E.3.2 Physical Standby Forwarding Redo to a Logical Standby

In this scenario, you have a primary database in a city in the United States and you wish to deploy three complete replicas of this database to be used for end-user queries and reporting in three different manufacturing plants in Europe. Your objective is to eliminate the need for users and applications at your European locations to access data that resides in the US to prevent network disruptions from making data unavailable for local access. While you can accept some latency between the time an update is made in the primary and the time it is replicated to all three european sites, you desire the data to be as up-to-date as possible and available to query and to run reports. You require a solution that is completely application transparent, and one where additional replicas can be deployed to sites in Europe if the need arises. A final requirement is the need to make this work with the limited bandwidth and very high network latency of the network connection between your US and European facilities.

You address your requirements by first creating a physical standby database in Europe for the primary database located in the US. You then create three logical standby databases, one in each of your European plants, and define each logical standby as a cascaded destination on your physical standby database. One copy of the redo is shipped over the transatlantic link from the US to the physical standby in Europe. The physical standby in Europe forwards the redo to the three logical standby databases in the European manufacturing plants providing local access to corporate data for end-user query and reports. Room for future growth is built in. Additional standby databases can be deployed in Europe without any modification to applications, without any additional overhead on your primary system, and without consuming any additional transatlantic bandwidth.

Configure the physical standby database to forward redo data to the logical standby databases in each of your manufacturing sites as in the example above. The only difference from the parameters in Example E–1 is that you will define two additional `LOG_ARCHIVE_DEST_`*n* parameters on the physical standby so that redo will be forwarded to all three logical standby databases.

F

Creating a Standby Database with Recovery Manager

This appendix describes how to use Oracle Recovery Manager to create a standby database. This appendix contains the following topics:

- Prerequisites
- Overview of Standby Database Creation with RMAN
- Using the DUPLICATE Command to Create a Standby Database

F.1 Prerequisites

This appendix assumes that you have read the chapter on database duplication in *Oracle Database Backup and Recovery User's Guide*. Because you use the `DUPLICATE` command to create a standby database with RMAN, you should familiarize yourself with the `DUPLICATE` command entry in *Oracle Database Backup and Recovery Reference*.

Familiarize yourself with how to create a standby database in Chapter 3, "Creating a Physical Standby Database" and Chapter 4, "Creating a Logical Standby Database" *before* you attempt the RMAN creation procedures described in this chapter.

F.2 Overview of Standby Database Creation with RMAN

This section explains the purpose and basic concepts involved in standby database creation with RMAN.

F.2.1 Purpose of Standby Database Creation with RMAN

You can use either manual techniques or the RMAN `DUPLICATE` command to create a standby database from backups of your primary database. Creating a standby database with RMAN has the following advantages over manual techniques:

- RMAN can create a standby database by copying the files currently in use by the primary database. No backups are required.
- RMAN can create a standby database by restoring backups of the primary database to the standby site. Thus, the primary database is not affected during the creation of the standby database.
- RMAN automates renaming of files, including Oracle Managed Files (OMF) and directory structures.
- RMAN restores archived redo log files from backups and performs media recovery so that the standby and primary databases are synchronized.

F.2.2 Basic Concepts of Standby Creation with RMAN

The procedure for creating a standby database with RMAN is almost the same as for creating a duplicate database. You need to amend the duplication procedures described in *Oracle Database Backup and Recovery User's Guide* to account for issues specific to a standby database.

To create a standby database with the `DUPLICATE` command you must connect as target to the primary database and specify the `FOR STANDBY` option. You cannot connect to a standby database and create an additional standby database. RMAN creates the standby database by restoring and mounting a control file. RMAN can use an existing backup of the primary database control file, so you do not need to create a control file backup especially for the standby database.

A standby database, unlike a duplicate database created by `DUPLICATE` *without* the `FOR STANDBY OPTION`, does not get a new DBID. Thus, you should not register the standby database with your recovery catalog.

F.2.2.1 Active Database and Backup-Based Duplication

You must choose between active and backup-based duplication. If you specify `FROM ACTIVE DATABASE`, then RMAN copies the datafiles directly from the primary database to the standby database. The primary database must be mounted or open.

If you not specify `FROM ACTIVE DATABASE`, then RMAN performs backup-based duplication. RMAN restores backups of the primary datafiles to the standby database. All backups and archived redo log files needed for creating and recovering the standby database must be accessible by the server session on the standby host. RMAN restores the most recent datafiles unless you execute the `SET UNTIL` command.

F.2.2.2 DB_UNIQUE_NAME Values in an RMAN Environment

A standby database, unlike a duplicate database created by `DUPLICATE` without the `FOR STANDBY` option, does not get a new DBID. When using RMAN in a Data Guard environment, you should always connect it to a recovery catalog. The recovery catalog can store the metadata for all primary and standby databases in the environment. You should not explicitly register the standby database in the recovery catalog.

A database in a Data Guard environment is uniquely identified by means of the `DB_UNIQUE_NAME` parameter in the initialization parameter file. The `DB_UNIQUE_NAME` must be unique across all the databases with the same DBID for RMAN to work correctly in a Data Guard environment.

> **See Also:** *Oracle Database Backup and Recovery User's Guide* for a conceptual overview of RMAN operation in a Data Guard environment

F.2.2.3 Recovery of a Standby Database

By default, RMAN does not recover the standby database after creating it. 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.

If you want RMAN to recover the standby database after creating it, then the standby control file must be usable for the recovery. The following 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 backing up the control file on the primary database. 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.

Use the `DORECOVER` option of the `DUPLICATE` command to specify that RMAN should recover the standby database. RMAN performs the following steps after creating the standby database files:

1. RMAN begins media recovery. If recovery requires archived redo log files, and if the log files are not already on disk, then RMAN attempts to restore 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.

F.2.2.4 Standby Database Redo Log Files

RMAN automatically creates the standby redo log files on the standby database. After the log files are created, the standby database maintains and archives them according to the normal rules for log files.

If you use backup-based duplication, then the only option when naming the standby redo log files on the standby database is the file names for the log files, as specified in the standby control file. If the log file names on the standby must be different from the primary file names, then one option is to specify file names for the standby redo logs by setting `LOG_FILE_NAME_CONVERT` in the standby initialization parameter file.

Note the following restrictions when specifying file names for the standby redo log files on the standby database:

- You must use the `LOG_FILE_NAME_CONVERT` parameter to name the standby 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 standby redo log files.
- You cannot use the `LOGFILE` clause of the `DUPLICATE` command to specify file names for the standby redo log files.
- If you want the standby redo log file names on the standby database to be the same as the primary redo log file names, then you must specify the `NOFILENAMECHECK` clause of the `DUPLICATE` command. Otherwise, RMAN signals an error even if the standby database is created on a different host.

F.2.2.5 Password Files for the Standby Database

If you are using active database duplication, then RMAN always copies the password file to the standby host because the password file on the standby database must be an exact copy of the password file on the target database. In this case, the `PASSWORD FILE` clause is not necessary. RMAN overwrites any existing password file for the auxiliary instance. With backup-based duplication you must copy the password file used on the primary to the standby, for Data Guard to ship logs.

F.3 Using the DUPLICATE Command to Create a Standby Database

The procedure for creating a standby database is basically identical to the duplication procedure described in *Oracle Database Backup and Recovery User's Guide*.

F.3.1 Creating a Standby Database with Active Database Duplication

To create a standby database from files that are active in the primary database, specify both `FOR STANDBY` and `FROM ACTIVE DATABASE`. Optionally, specify the `DORECOVER` option to recover the database after standby creation.

This scenario assumes that the standby host and primary database host have the same directory structure.

To create a standby database from active database files:

1. Prepare the auxiliary database instance as explained in *Oracle Database Backup and Recovery User's Guide*.

   Because you are using active database duplication, you must create a password file for the auxiliary instance and establish Oracle Net connectivity. This is a temporary password file as it will be overwritten during the duplicate operation.

2. Decide how to provide names for the standby control files, datafiles, online redo logs, and tempfiles. This step is explained in *Oracle Database Backup and Recovery User's Guide*.

   In this scenario, the standby database files will be named the same as the primary database files.

3. Start and configure RMAN as explained in *Oracle Database Backup and Recovery User's Guide*.

4. Execute the `DUPLICATE` command.

   The following example illustrates how to use `DUPLICATE` for active duplication. This example requires the `NOFILENAMECHECK` option because the primary database files have the same names as the standby database files. The `SET` clauses for SPFILE are required for log shipping to work properly. The `db_unique_name` must be set to ensure that the catalog and Data Guard can identify this database as being different from the primary.

   ```
   DUPLICATE TARGET DATABASE
     FOR STANDBY
     FROM ACTIVE DATABASE
     DORECOVER
     SPFILE
       SET "db_unique_name"="foou" COMMENT ''Is a duplicate''
       SET LOG_ARCHIVE_DEST_2="service=inst3 ASYNC REGISTER
        VALID_FOR=(online_logfile,primary_role)"
       SET FAL_CLIENT="inst3" COMMENT "Is standby"
       SET FAL_SERVER="inst1" COMMENT "Is primary"
     NOFILENAMECHECK;
   ```

   RMAN automatically copies the server parameter file to the standby host, starts the auxiliary instance with the server parameter file, restores a backup control file, and copies all necessary database files and archived redo logs over the network to the standby host. RMAN recovers the standby database, but does not place it in manual or managed recovery mode.

F.3.2 Creating a Standby Database with Backup-Based Duplication

To create a standby database from backups, specify `FOR STANDBY` but do not specify `FROM ACTIVE DATABASE`. Optionally, specify the `DORECOVER` option to recover the database after standby creation.

This scenario assumes that the standby host and primary database host have the same directory structure.

To create a standby database from backups:

1. Make database backups and archived redo logs available to the auxiliary instance on the duplicate host as explained in *Oracle Database Backup and Recovery User's Guide*.
2. Prepare the auxiliary database instance as explained in *Oracle Database Backup and Recovery User's Guide*.
3. Decide how to provide names for the standby control files, datafiles, online redo logs, and tempfiles. This step is explained in *Oracle Database Backup and Recovery User's Guide*.

   In this scenario, the standby database files will be named the same as the primary database files.
4. Start and configure RMAN as explained in *Oracle Database Backup and Recovery User's Guide*.
5. Execute the `DUPLICATE` command.

   The following example illustrates how to use `DUPLICATE` for backup-based duplication. This example requires the `NOFILENAMECHECK` option because the primary database files have the same names as the standby database files.

   ```
   DUPLICATE TARGET DATABASE
     FOR STANDBY
     DORECOVER
     SPFILE
       SET "db_unique_name"="foou" COMMENT ''Is a duplicate''
       SET LOG_ARCHIVE_DEST_2="service=inst3 ASYNC REGISTER
        VALID_FOR=(online_logfile,primary_role)"
       SET FAL_CLIENT="inst3" COMMENT "Is standby"
       SET FAL_SERVER="inst1" COMMENT "Is primary"
     NOFILENAMECHECK;
   ```

   RMAN automatically copies the server parameter file to the standby host, starts the auxiliary instance with the server parameter file, and restores all necessary database files and archived redo logs to the standby host. RMAN recovers the standby database, but does not place it in manual or managed recovery mode.

G

Setting Archive Tracing

The Oracle database uses the LOG_ARCHIVE_TRACE parameter to enable and control the generation of comprehensive trace information for log archiving and redo transport activity. This tracing information is written to the Automatic Diagnostic Repository.

This appendix contains the following sections:

- Setting the LOG_ARCHIVE_TRACE Initialization Parameter
- Choosing an Integer Value

> **See Also:** *Oracle Database Administrator's Guide* for more information about the Automatic Diagnostic Repository

G.1 Setting the LOG_ARCHIVE_TRACE Initialization Parameter

The format for the archiving trace parameter is as follows, where *trace_level* is an integer:

```
LOG_ARCHIVE_TRACE=trace_level
```

To enable, disable, or modify the LOG_ARCHIVE_TRACE parameter for a physical standby database, issue a SQL statement similar to the following:

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_TRACE=15;
```

In the previous example, setting the LOG_ARCHIVE_TRACE parameter to a value of 15 sets trace levels 1, 2, 4, and 8 as described in Section G.2.

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;
```

G.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

Level	Meaning
2	Tracks archive status by archive log file destination
4	Tracks archive operational phase
8	Tracks archive log destination activity
16	Tracks detailed archive log destination activity
32	Tracks archive log destination parameter modifications
64	Tracks ARC*n* process state activity
128	Tracks FAL server process activity
256	Tracks RFS Logical Client
512	Tracks LGWR redo shipping network activity
1024	Tracks RFS physical client
2048	Tracks RFS/ARC*n* ping heartbeat
4096	Tracks real-time apply activity
8192	Tracks Redo Apply activity (media recovery or physical standby)

You can combine tracing levels by setting the value of the `LOG_ARCHIVE_TRACE` parameter to the sum of the individual levels. For example, setting the parameter to 6 generates level 2 and level 4 trace output.

The following are examples of the ARC0 trace data generated on the primary site by the archiving of log file 387 to two different destinations: the service `standby1` and the local directory `/oracle/dbs`.

Note: The level numbers do not appear in the actual trace output; they are shown here for clarification only.

```
Level   Corresponding entry content (sample)
-----   -------------------------------
( 1)    ARC0: Begin archiving log# 1 seq# 387 thrd# 1
( 4)    ARC0: VALIDATE
( 4)    ARC0: PREPARE
( 4)    ARC0: INITIALIZE
( 4)    ARC0: SPOOL
( 8)    ARC0: Creating archive destination 2 : 'standby1'
(16)    ARC0:  Issuing standby Create archive destination at 'standby1'
( 8)    ARC0: Creating archive destination 1 : '/oracle/dbs/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
```

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z