Oracle® Data Guard

Concepts and Administration

12*c* Release 2 (12.2)
E85767-01
May 2017

Oracle Data Guard Concepts and Administration, 12*c* Release 2 (12.2)

E85767-01

Primary Author: Kathy Rich

Contributors: Andy Adams, Beldalker Anand, Chipper Brown, Larry Carpenter, Jin-Jwei Chen, Laurence Clarke, Jeff Detjen, Ray Dutcher, David Gagne, B.G. Garin, Mahesh Girkar, Yuhong Gu, Joydip Kundu, Steven Lee, Steven Lim, Nitin Karkhanis, Goutam Kulkarni, Jonghyun Lee, Yunrui Li, Shashi Mangalat, Steven McGee, Bob McGuirk, Joe Meeks, Steve Moriarty, Muthu Olagappan, Ashish Ray, Mike Schloss, Mike Smith, Lawrence To, Stephen Vivian, Doug Voss, Hongjie Yang

Contents

11 Managing a Logical Standby Database

12 Using RMAN to Back Up and Restore Files

15 Oracle Data Guard Scenarios

Part II Reference

16 Initialization Parameters

B Patching, Upgrading, and Downgrading Databases in an Oracle Data Guard Configuration

C Data Type and DDL Support on a Logical Standby Database

D Oracle Data Guard and Oracle Real Application Clusters

E Creating a Standby Database with Recovery Manager

F Setting Archive Tracing

G Performing Role Transitions Using Old Syntax

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

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

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 Oracle Data Guard configurations.

- *Oracle Database High Availability Overview* for information about how Oracle Data Guard is used as a key component in high availability and disaster recovery environments.
- Oracle Enterprise Manager Cloud Control online Help system

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.

Changes in This Release for Oracle Data Guard Concepts and Administration

This preface lists changes in *Oracle Data Guard Concepts and Administration*.

Changes in Oracle Database 12c Release 2 (12.2.0.1)

The following are changes in *Oracle Data Guard Concepts and Administration* for Oracle Database 12*c* Release 2 (12.2.0.1).

- A new `INSTANCES [ ALL | integer]` clause is available on the SQL `ALTER RECOVER MANAGED STANDBY DATABASE` command which enables you to control the number of instances on a physical standby that Redo Apply uses. See Setting Up Multi-Instance Redo Apply.
- Logical standby now supports long identifiers (128 bytes).
- You can now upgrade databases that use Oracle Label Security (OLS) to new Oracle Database releases and patch sets by using Oracle Data Guard database rolling upgrades using a transient logical standby database and the PL/SQL package, `DBMS_ROLLING`. See Oracle Label Security.
- You can now upgrade databases that use Oracle Database Vault to new Oracle Database releases and patch sets by using Oracle Data Guard database rolling upgrades with a transient logical standby and the PL/SQL package, `DBMS_ROLLING`. See Oracle Database Vault.
- A new database initialization parameter, `DATA_GUARD_SYNC_LATENCY`, enables you to define the maximum amount of time (in seconds) that the primary database may wait before disconnecting subsequent destinations after at least one synchronous standby has acknowledged receipt of the redo. See Configuring an Oracle Database to Send Redo Data
- You can now detect lost writes and also inconsistencies between a primary database and physical standby databases by using the new PL/SQL procedure, `DBMS_DBCOMP.DBCOMP`. See Using the DBCOMP Procedure to Detect Lost Writes and Other Inconsistencies.
- The new `ENABLED_PDBS_ON_STANDBY` initialization parameter enables you to specify a subset of pluggable databases (PDBs) for replication on a physical standby of a multitenant container database (CDB). In releases prior to Oracle Database 12*c* Release 2 (12.2.0.1), you had to specify either all PDBs or none. See Creating a Physical Standby of a CDB.
- Oracle Database In-Memory column store (IM column store) is now supported on standby databases in Oracle Active Data Guard (ADG) environments. See IM Column Store in an Active Data Guard Environment.
- Oracle Active Data Guard is integrated with Oracle Sharding. See Oracle Active Data Guard Supports Oracle Sharding.

- You can use the new `FARSYNC` option on the RMAN `DUPLICATE` command to create an Oracle Data Guard far sync instance. You can do so using either active database duplication or backup-based duplication. See Using the DUPLICATE Command to Create a Standby Database.
- You can now use the Oracle Diagnostic Pack with an Oracle Active Data Guard standby database that is open read-only. See Using Oracle Diagnostic Pack to Tune Oracle Active Data Guard Standbys.
- The number of alternate log archive destinations that you can define has been increased with the capability to create groups of log archive destinations. You can define priority of group members, as well as policies in a failure state. See Alternate Destinations.
- Rolling upgrades performed using the `DBMS_ROLLING` PL/SQL package are supported on multitenant container databases (CDBs). See DBMS_ROLLING Upgrades and CDBs.
- Password file changes done on the primary database are now automatically propagated to standby databases. The only exception to this is far sync instances. Updated password files must still be manually copied to far sync instances because far sync instances receive redo, but do not apply it. Once the password file is up-to-date at the far sync instance, the redo containing the password update at the primary is automatically propagated to any standby databases that are set up to receive redo from that far sync instance. The password file is updated on the standby when the redo is applied.
- When a physical standby database is converted into a primary, there is now an option to keep any sessions connected to the standby during the switchover/ failover. This is done with the `STANDBY_DB_PRESERVE_STATES` initialization parameter. See Role Transitions Involving Physical Standby Databases.
- You can encrypt and decrypt both new and existing tablespaces, and existing databases within an Oracle Data Guard environment. This can be done offline or online. See Reset the TDE Master Encryption Key and Support for Tablespace Encryption.

Part I
Concepts and Administration

The following topics provide information about Oracle Data Guard concepts and administration:

1

Introduction to Oracle Data Guard

Oracle Data Guard ensures high availability, data protection, and disaster recovery for enterprise data.

Oracle 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. Oracle 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, Oracle Data Guard can switch any standby database to the production role, minimizing the downtime associated with the outage. Oracle Data Guard can be used with traditional backup, restoration, and cluster techniques to provide a high level of data protection and data availability. Oracle Data Guard transport services are also used by other Oracle features such as Oracle Streams and Oracle GoldenGate for efficient and reliable transmission of redo from a source database to one or more remote destinations.

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

See the following topics which describe the highlights of Oracle Data Guard:

1.1 Oracle Data Guard Configurations

An **Oracle Data Guard configuration** can contain one primary database and up to thirty destinations.

The members of an Oracle Data Guard configuration are connected by Oracle Net and may be dispersed geographically. There are no restrictions on where the members of an Oracle Data Guard configuration are located as long as they can communicate with each other. For example, you can have a standby database in the same data center as the primary database, along with two standbys in another data center.

You can manage primary and standby databases using either the SQL command-line interface or the Oracle Data Guard broker interfaces. The broker provides a command-

line interface (DGMGRL) and a graphical user interface that is integrated in Oracle Enterprise Manager Cloud Control.

1.1.1 Primary Database

An Oracle Data Guard configuration contains one production database, also referred to as the primary database, that functions in the primary role.

The primary database 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 (Oracle 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 thirty standby databases and incorporate them into an Oracle Data Guard configuration. Oracle 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.

  Additionally, as of Oracle Database 11*g* Release 2 (11.2.0.1), a physical standby database can be used to install eligible one-off patches, patch set updates (PSUs), and critical patch updates (CPUs), in rolling fashion. For more information about this functionality, see the My Oracle Support note 1265700.1 at `http://support.oracle.com`.

- **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.

  The flexibility of a logical standby database lets you upgrade Oracle Database software (patch sets and new Oracle Database releases) and perform other database maintenance in rolling fashion with almost no downtime. From Oracle

Database 11*g* onward, the transient logical database rolling upgrade process can also be used with existing physical standby databases.

- **Snapshot Standby Database**

  A snapshot standby database is a fully updatable 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. For example, you can use the Oracle Real Application Testing option to capture the database workload on a primary and then replay it for test purposes on the snapshot standby. 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.

> **See Also:**
>
> - *Oracle Database Testing Guide* for more information about Oracle Real Application Testing and the license required to use it

1.1.3 Far Sync Instances

An Oracle Data Guard far sync instance is a remote Oracle Data Guard destination that accepts redo from the primary database and then ships that redo to other members of the Oracle Data Guard configuration.

A far sync instance manages a control file, receives redo into standby redo logs (SRLs), and archives those SRLs to local archived redo logs, but that is where the similarity with standbys ends. A far sync instance does not have user data files, cannot be opened for access, cannot run redo apply, and can never function in the primary role or be converted to any type of standby database.

Far sync instances are part of the Oracle Active Data Guard Far Sync feature, which requires an Oracle Active Data Guard license.

> **See Also:**
>
> - Far Sync

1.1.4 Zero Data Loss Recovery Appliance

Zero Data Loss Recovery Appliance (Recovery Appliance) is an enterprise-level backup solution that provides a single repository for backups of all of your Oracle databases.

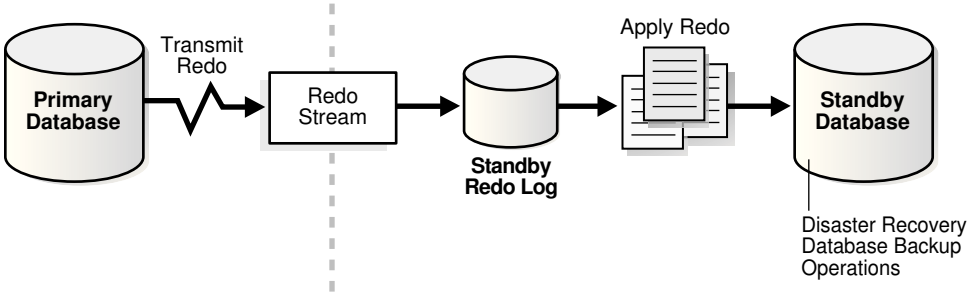
Recovery Appliance offloads most Oracle Database backup and restore processing to a centralized backup system. It enables you to achieve significant efficiencies in storage utilization, performance, and manageability of backups.

> **See Also:**
>
> - *Zero Data Loss Recovery Appliance Administrator's Guide*

1.1.5 Configuration Example

Figure 1-1 shows a typical Oracle 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 Oracle Data Guard Configuration



1.2 Oracle Data Guard Services

Oracle Data Guard uses Redo Transport Services and Apply Services to manage 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

Redo data is applied directly from standby redo log files as they are filled using real-time apply. If standby redo log files are not configured, then redo data must first be archived at the standby database before it is applied.

- 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

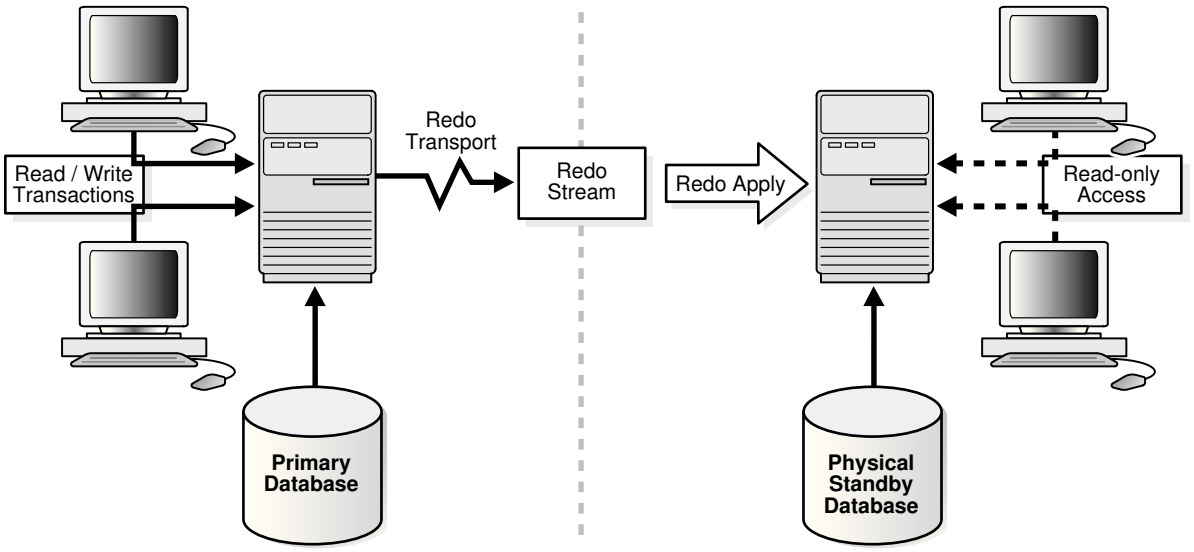
Apply services automatically apply the redo data on the standby database to maintain consistency with the primary database.

The redo data is transmitted from the primary database and written to the standby redo log on the standby database. Apply services 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:

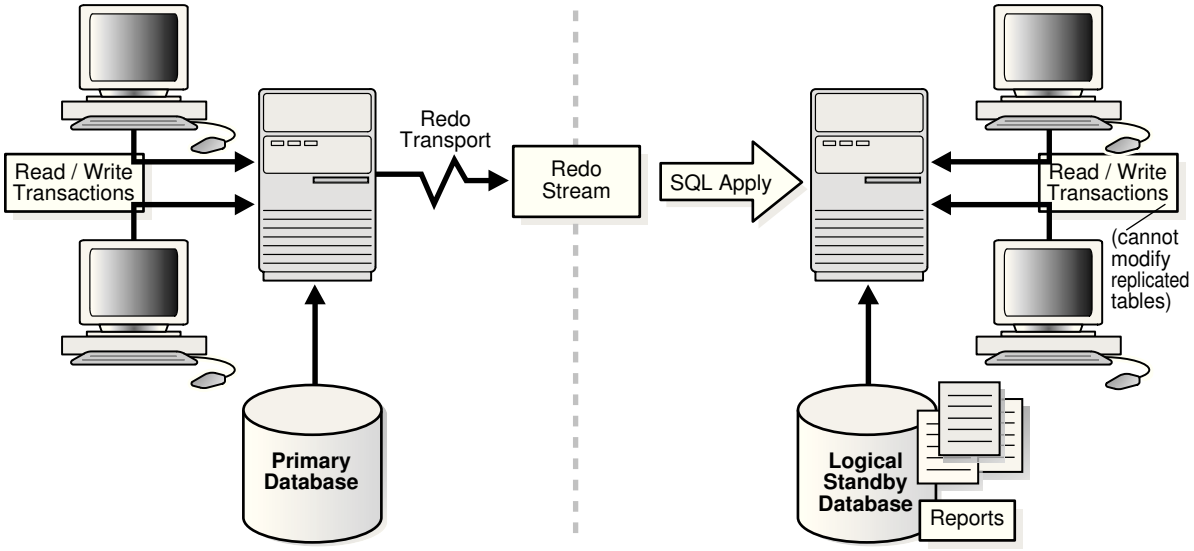
- For physical standby databases, Oracle Data Guard uses **Redo Apply** technology, which applies redo data on the standby database using standard recovery techniques of an Oracle database, as shown in Figure 1-2.

Figure 1-2 Automatic Updating of a Physical Standby Database



- For logical standby databases, Oracle 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

Using Oracle Data Guard, you can change the role of a database using either a switchover or a failover operation.

An Oracle database operates in one of two roles: primary or standby.

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 Oracle 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 Cloud Control or the DGMGRL command-line interface, as described in Oracle Data Guard Broker.

1.3 Oracle Data Guard Broker

The Oracle Data Guard broker is a distributed management framework that automates the creation, maintenance, and monitoring of Oracle Data Guard configurations.

You can use either the Oracle Enterprise Manager Cloud Control graphical user interface (GUI) or the Oracle Data Guard command-line interface (DGMGRL) to:

- Create and enable Oracle Data Guard configurations, including setting up redo transport services and apply services
- Manage an entire Oracle Data Guard configuration from any system in the configuration
- Manage and monitor Oracle 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 Cloud Control or a single command in the DGMGRL command-line interface.
- Enable Oracle Data Guard fast-start failover to fail over *automatically* when the primary database becomes unavailable. When fast-start failover is enabled, the Oracle 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 Cloud Control 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 Oracle 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 Cloud Control

Oracle Enterprise Manager Cloud Control provides a web-based interface for viewing, monitoring, and administering primary and standby databases in an Oracle Data Guard configuration.

Oracle Enterprise Manager Cloud Control is sometimes referred to simply as Cloud Control.

Enterprise Manager's easy-to-use interfaces, combined with the broker's centralized management and monitoring of the Oracle Data Guard configuration, enhance the Oracle Data Guard solution for high availability, site protection, and data protection of an enterprise.

Using Enterprise Manager, 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.

1.3.2 Using the Oracle Data Guard Command-Line Interface

The Oracle Data Guard command-line interface (DGMGRL) enables you to control and monitor an Oracle 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 Oracle Data Guard Protection Modes

Oracle Data Guard provides three distinct modes of data protection.

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 fails. The following are brief descriptions of the protection modes available for each of these situations:

Maximum Availability

This protection mode provides the highest level of data protection that is possible without compromising the availability of a primary database. With Oracle Data Guard, transactions do not commit until all redo data needed to recover those transactions has either been received in memory or written to the standby redo log (depending upon configuration) on 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 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 occurs 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 the standby redo log on at least one synchronized standby database before the transaction commits. To ensure that data loss cannot occur, the primary database shuts down, rather than continue processing transactions, if it cannot write its redo stream to at least one synchronized 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 Also:**
>
> - Oracle Data Guard Protection Modes for more detailed descriptions of these modes and for 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.

> **See Also:**
>
> - *Oracle Data Guard Broker* for information about configuration requirements specific to Oracle Data Guard for Fast Application Notification (FAN), Fast Connection Failover (FCF), and role-specific database services
> - The Maximum Availability Architecture client failover best practices white paper at
>
>   `http://www.oracle.com/goto/maa`

1.5.1 Application Continuity

Application Continuity is an Oracle Database feature that enables rapid and nondisruptive replays of requests against the database after a recoverable error that made the database session unavailable.

Application Continuity is supported for Oracle Data Guard switchovers to physical standby databases. It is also supported for fast-start failover to physical standbys in maximum availability data protection mode. To use Application Continuity, the primary and standby databases must be licensed for Oracle Real Application Clusters (Oracle RAC) or Oracle Active Data Guard.

> **See Also:**
>
> - *Oracle Real Application Clusters Administration and Deployment Guide* for information about Application Continuity

1.6 Oracle Data Guard and Complementary Technologies

Oracle Database provides several unique technologies that complement Oracle 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 (Oracle 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 Oracle 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.

  - Oracle 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 Oracle 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 Oracle Data Guard and Oracle Real Application Clusters and *Oracle Database High Availability Overview* for Oracle RAC and Oracle Data Guard integration.

- Oracle Real Application Clusters One Node (Oracle RAC One Node)

  Oracle RAC One Node provides enhanced high availability for noncluster databases, protecting them from both planned and unplanned downtime. Oracle RAC One Node provides the following:

  - Always-on noncluster database services
  - Better consolidation for database servers
  - Enhanced server virtualization
  - Lower cost development and test platform for full Oracle RAC

  In addition, Oracle RAC One Node facilitates the consolidation of database storage, standardizes your database environment, and, when necessary, enables you to upgrade to a full, multinode Oracle RAC database without downtime or disruption.

  As of Oracle Database 11*g* Release 2 (11.2.0.2), Oracle Data Guard and Oracle Data Guard broker are fully integrated with Oracle Real Application Clusters One Node (Oracle RAC One Node).

- 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 data files.
  - 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 Specifying a Time Delay for the Application of Archived Redo Log Files 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 Oracle Data Guard, RMAN is a feature of the Oracle database and does not require separate installation. Oracle 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 Creating a Standby Database with Recovery Manager.

- Oracle Global Data Services (GDS)

  Oracle Global Data Services (GDS) applies the Oracle RAC service model to pools of globally distributed databases, providing dynamic load balancing, failover, and centralized service management for a set of replicated databases that offer common services. The set of databases can include Oracle RAC and single-instance Oracle databases interconnected through Oracle Data Guard, Oracle GoldenGate, or any other replication technology.

  GDS is integrated with Oracle Data Guard broker. This allows role-specific global services to be automatically started and stopped as appropriate when role transitions occur within an Oracle Data Guard broker configuration.

  GDS allows the specification of a replication lag limit for a global service. If the lag limit is exceeded at a given replica, the global service is temporarily stopped at that replica and new client requests are routed to a replica that satisfies the lag limit. The global service is automatically restarted at the original replica when the replication lag becomes less than the lag limit.

1.7 Oracle Active Data Guard Supports Oracle Sharding

Oracle Sharding allows you to horizontally partition data across multiple independent Oracle databases and route database connection requests to databases that contain appropriate data. Oracle Data Guard and Oracle Sharding are integrated technologies.

Sharding splits data into multiple independent databases (shards) that do not share any physical resources. Sharding is usually combined with data replication, such as that provided by Oracle Data Guard. Oracle Data Guard provides fast single-master replication of an entire Oracle Database. In an Oracle Data Guard configuration there is an updateable primary database and one or more standby databases which can be open for read-only access. Replication can improve performance and scalability of a sharded database and provide high-availability and disaster recovery. Replication topology in a sharded database is specified using the `-repl` option on the GDSCTL `create shardcatalog` command. The default replication topology is Oracle Data Guard.

Oracle Sharding supports two methods of sharding: system-managed and composite. The sharding method is specified with the GDSCTL command `CREATE SHARDCATALOG`.

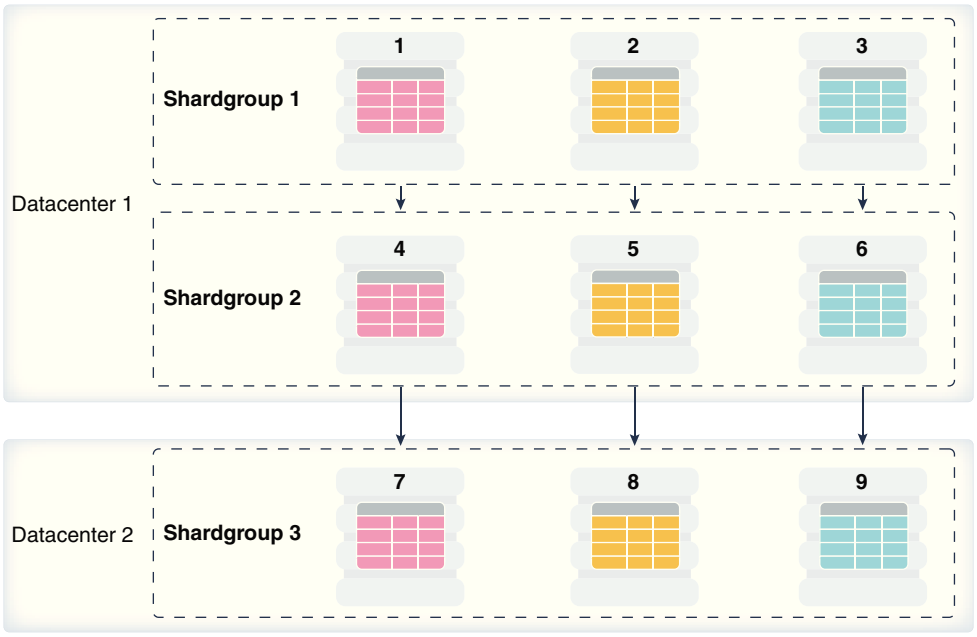
Shards that belong to a shardgroup are usually located in the same data center. An entire shardgroup can be fully replicated to one or more shardgroups in the same or different data centers.

System-Managed Sharding With Oracle Data Guard Replication

In system-managed sharding, the logical unit of replication is a group of shards called shardgroup. In system-managed sharding, a shardgroup contains all data stored in the

sharded database. The data is automatically distributed across shards using partitioning by consistent hash (consistent hash is a partitioning strategy commonly used in scalable distributed systems). The partitioning algorithm evenly and randomly distributes data across shards. The following figure illustrates how Oracle Data Guard replication is used with system-managed sharding. There is a primary shardgroup – Shardgroup 1 and two standby shardgroups - Shardgroup 2 and Shardgroup 3.

Figure 1-4 System-Managed Sharding With Oracle Data Guard Replication



Shardgroup 1 consists of Oracle Data Guard primary databases (shards 1, 2, 3). Shardgroup 2 consists of local standby databases (shards 4, 5, 6) which are located in the same datacenter and configured for synchronous replication. And Shardgroup 3 consists of remote standbys (shards 7, 8, 9) located in a different datacenter and configured for asynchronous replication. The default replication topology is Oracle Data Guard. To open each standby in the configuration in read-only mode, you must enable Oracle Active Data Guard. This is done using the `-deploy_as ACTIVE_STANDBY` option on the GDSCTL `add shardgroup` command.

The sharded database shown in the previous figure consists of three sets of replicated shards: {1, 4, 7}, {2, 5, 8} and {3, 6, 9}. Each set of replicated shards is managed as an Oracle Data Guard broker configuration with fast-start failover enabled.

To deploy replication, you only need to specify the properties of shardgroups (region, role, etc) and add shards to them. Oracle Sharding automatically configures Oracle Data Guard and starts a fast-start failover observer for each set of replicated shards. It also provides load balancing of read-only workloads, role-based global services, and replication lag and locality-based routing.

To deploy the configuration shown in the previous figure, execute the following GDSCTL commands:

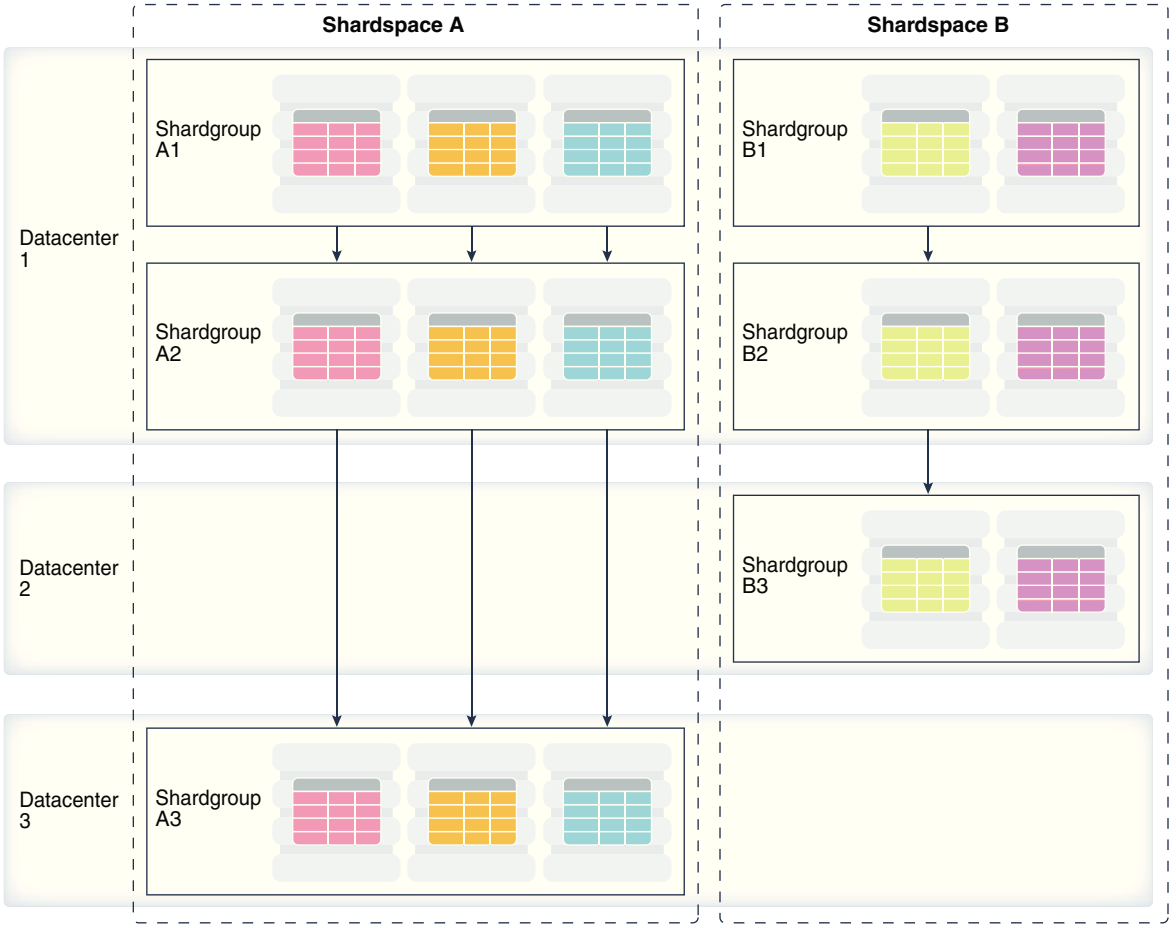
```
CREATE SHARDCATALOG –database host00:1521:shardcat –region dc1, dc2
```

```
ADD GSM -gsm gsm1 -listener 1571 –catalog host00:1521:shardcat –region dc1
ADD GSM -gsm gsm2 -listener 1571 –catalog host00:1521:shardcat –region dc2

ADD SHARDGROUP -shardgroup shardgroup1 -region dc1 -deploy_as primary
ADD SHARDGROUP -shardgroup shardgroup2 -region dc1 -deploy_as standby
ADD SHARDGROUP -shardgroup shardgroup3 -region dc2 -deploy_as standby

CREATE SHARD -shardgroup shardgroup1 -destination host01 -credential oracle_cred  -
netparamfile /home/oracle/netca_dbhome.rsp
CREATE SHARD -shardgroup shardgroup1 -destination host02 -credential oracle_cred  -
netparamfile /home/oracle/netca_dbhome.rsp
CREATE SHARD -shardgroup shardgroup1 -destination host03 -credential oracle_cred  -
netparamfile /home/oracle/netca_dbhome.rsp
……
CREATE SHARD -shardgroup shardgroup3  -destination host09 -credential oracle_cred -
netparamfile /home/oracle/netca_dbhome.rsp

DEPLOY
```

Composite Sharding With Oracle Data Guard Replication

Composite sharding combines features of system-managed sharding and user-managed sharding. In composite sharding the logical unit of replication is a group of shards called a shardgroup (as in system-managed sharding). Also in composite sharding, a sharded database consists of multiple shardspaces (as in user-managed sharding), however, each shardspace, instead of replicated shards, contains replicated shardgroups as shown in the following figure.

Figure 1-5 Composite Sharding With Oracle Data Guard Replication



To deploy the configuration shown in the previous figure, execute the following GDSCTL commands:

```
CREATE SHARDCATALOG -sharding composite –database host00:1521:cat –region dc1, dc2,
dc3

ADD GSM -gsm gsm1 -listener 1571 –catalog host00:1521:cat –region dc1
ADD GSM -gsm gsm2 -listener 1571 –catalog host00:1521:cat –region dc2
ADD GSM -gsm gsm3 -listener 1571 –catalog host00:1521:cat –region dc3

ADD SHARDSPACE -shardspace shardspace_a
ADD SHARDSPACE -shardspace shardspace_b

ADD SHARDGROUP -shardgroup shardgroup_a1 –shardspace shardspace_a -region dc1
-deploy_as primary
ADD SHARDGROUP -shardgroup shardgroup_a2 –shardspace shardspace_a -region dc1     -
deploy_as standby
ADD SHARDGROUP -shardgroup shardgroup_a3 –shardspace shardspace_a -region dc3     -
deploy_as standby
ADD SHARDGROUP -shardgroup shardgroup_b1 –shardspace shardspace_a -region dc1
-deploy_as primary
ADD SHARDGROUP -shardgroup shardgroup_b2 –shardspace shardspace_a -region dc1     -
deploy_as standby
ADD SHARDGROUP -shardgroup shardgroup_b3 –shardspace shardspace_a -region dc2     -
```

```
deploy_as standby

CREATE SHARD -shardgroup shardgroup_a1 -destination host01 –credential orcl_cred  -
netparamfile /home/oracle/netca_dbhome.rsp
……
CREATE SHARD -shardgroup shardgroup_b3 -destination host09 -credential
orcl_cred      -netparamfile /home/oracle/netca_dbhome.rsp

DEPLOY
```

Related Topics:

- *Oracle Database Administrator's Guide*
- *Oracle Database Global Data Services Concepts and Administration Guide*

1.8 Summary of Oracle Data Guard Benefits

Oracle Data Guard provides an efficient and comprehensive disaster recovery and high availability solution.

Oracle Data Guard offers these benefits:

- High availability

  Oracle Data Guard's 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

  Oracle Data Guard can ensure zero data loss, even in the face of unforeseen disasters. A standby database provides a safeguard against unplanned outages of all types, including data corruption and administrative error. Because the redo data received from a primary database is validated at a standby database, physical corruptions that can occur at a primary database are not propagated to the standby database. Additional validation performed at a standby database also prevents logical intra-block corruptions and lost-write corruptions from propagating to the standby. Similarly, administrative errors such as accidental file deletions by a storage administrator are not propagated to a standby database. A physical standby database can also be used to protect against user errors either by delaying the redo apply or by using Flashback Database to rewind the standby and extract a good copy of the data.

- 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), then redo data being generated on the primary database cannot be sent to those standby databases. After a connection is reestablished, the missing archived redo log files (referred to as a gap) are automatically detected by Oracle 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 Oracle Data Guard broker provides a graphical user interface and a command-line interface to automate management and operational tasks across multiple databases in an Oracle Data Guard configuration. The broker also monitors all of the systems within a single Oracle Data Guard configuration.

- Integration with Oracle Database

  Oracle 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 Oracle 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 Oracle Data Guard

The information in this section describes how to get started using Oracle Data Guard.

See the following topics:

- Standby Database Types
- User Interfaces for Administering Oracle Data Guard Configurations
- Oracle Data Guard Operational Prerequisites
- Standby Database Directory Structure Considerations
- Moving the Location of Online Data Files

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, Oracle 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. An Oracle 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.

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, Redo Apply can be active while the physical standby database is open, thus allowing queries to return results that are identical to what would be returned from the primary database. This capability is known as the real-time query feature.

> **See Also:**
>
> - "Opening a Physical Standby Database "
> - *Oracle Database Licensing Information* for more information about Oracle Active Data Guard

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 are not 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. The flexibility of a logical standby database lets you upgrade Oracle Database software (patch sets and new Oracle Database releases) and perform other database maintenance in rolling fashion with almost no downtime. From Oracle Database 11*g* onward, the transient logical database rolling upgrade process can also be used with existing physical standby databases.

Oracle 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.

A logical standby database has some restrictions on data types, types of tables, and types of DDL and DML operations. See Data Type and DDL Support on a Logical Standby Database 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:

- Minimizing downtime on software upgrades

  A logical standby database is ideal for upgrading an Oracle Data Guard configuration in a rolling fashion. 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. Logical standbys provide the underlying platform for the `DBMS_ROLLING` PL/SQL package, which provides functionality that allows you to make your Oracle Data Guard configuration highly available in the context of rolling upgrades and other storage reorganization.

- Support 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 can be used to drive the creation of data cubes and other OLAP data views. However, a logical standby database does not allow for any transformation of your data (such as replicating only a subset of columns or allowing additional columns on user tables). For those types of reporting activities, Oracle GoldenGate is Oracle's preferred solution.

2.1.3 Snapshot Standby Databases

A snapshot standby database is a type of updatable standby database that provides full data protection for a primary 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 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 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 is then 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 the 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. You can use the Oracle Real Application Testing option to capture primary database workload and then replay it for test purposes on the snapshot standby.
- 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.

> **See Also:**
>
> - *Oracle Database Testing Guide* for more information about Oracle Real Application Testing and the license required to use it

2.2 User Interfaces for Administering Oracle Data Guard Configurations

Oracle Data Guard provides several interfaces that you can use to configure, implement, and manage an Oracle Data Guard configuration.

- Oracle Enterprise Manager Cloud Control

  Oracle Enterprise Manager Cloud Control provides a GUI interface for the Oracle Data Guard broker that automates many of the tasks involved in creating, configuring, and monitoring an Oracle Data Guard environment. See the Oracle Enterprise Manager Cloud Control 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 SQL Statements Relevant to Oracle Data Guard for a list of the relevant statements.

- Initialization parameters

Several initialization parameters are used to define the Oracle Data Guard environment. See Initialization Parameters for a list of the relevant initialization parameters.

- Oracle Data Guard broker command-line interface (DGMGRL)

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

2.3 Oracle Data Guard Operational Prerequisites

The use of Oracle Data Guard requires certain hardware and software prerequisites.

- Hardware and Operating System Requirements
- Oracle Software Requirements

2.3.1 Hardware and Operating System Requirements

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 or transient logical standby databases.

As of Oracle Database 11*g*, Oracle Data Guard provides increased flexibility for Oracle Data Guard configurations in which the primary and standby systems may have different CPU architectures, operating systems (for example, Windows and Linux), operating system binaries (32-bit/64-bit), or Oracle database binaries (32-bit/64-bit).

This increased mixed-platform flexibility is subject to the current restrictions documented in the My Oracle Support notes 413484.1 and 1085687.1 at `http://support.oracle.com`.

Note 413484.1 discusses mixed-platform support and restrictions for physical standbys.

Note 1085687.1 discusses mixed-platform support and restrictions for logical standbys.

> **See Also:**
>
> - Using SQL Apply to Upgrade the Oracle Database for information about rolling database upgrades

2.3.2 Oracle Software Requirements

To use Oracle Data Guard, you must meet certain Oracle software requirements.

- 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, registering them, and using media recovery to roll forward the copy of the database at the disaster recovery site. Such a configuration does not provide the ease-of-use, manageability, performance, and disaster-recovery capabilities available with Oracle Data Guard.

- Using Oracle Data Guard SQL Apply, you can 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 Using SQL Apply to Upgrade the Oracle Database and the ReadMe file for the applicable Oracle Database 10*g* patch set release.
- The `COMPATIBLE` database initialization parameter must be set to the same value on all databases in an Oracle Data Guard configuration, except when using a logical standby database, which can have a higher `COMPATIBLE` setting than the primary database.
- 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 Clusters (Oracle 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.
- 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 data file 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 either the `SYSDG` or `SYSDBA` administrative privilege.
- For operational simplicity, Oracle recommends that when you set up Oracle Automatic Storage Management (Oracle ASM) and Oracle Managed Files (OMF) in an Oracle Data Guard configuration that you set it up symmetrically on the primary and standby database(s). If any database in the Oracle Data Guard configuration uses Oracle ASM, OMF, or both, then every database in the configuration should use Oracle ASM, OMF, or both, respectively, unless you are purposely implementing a mixed configuration for migration or maintenance purposes. See the scenario in Creating a Standby Database That Uses OMF or Oracle ASM 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 data files, archived redo log files, and standby redo log files.

If possible, the data files, 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 data file. 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:**
>
> For operational simplicity, Oracle recommends that when you set up Oracle Automatic Storage Management (Oracle ASM) and Oracle Managed Files (OMF) in an Oracle Data Guard configuration that you set it up symmetrically on the primary and standby database(s). If any database in the Oracle Data Guard configuration uses Oracle ASM, OMF, or both, then every database in the configuration should use Oracle ASM, OMF, or both, respectively, unless you are purposely implementing a mixed configuration for migration or maintenance purposes. See the scenario in Creating a Standby Database That Uses OMF or Oracle ASM for more information.

Figure 2-1 Possible Standby Configurations

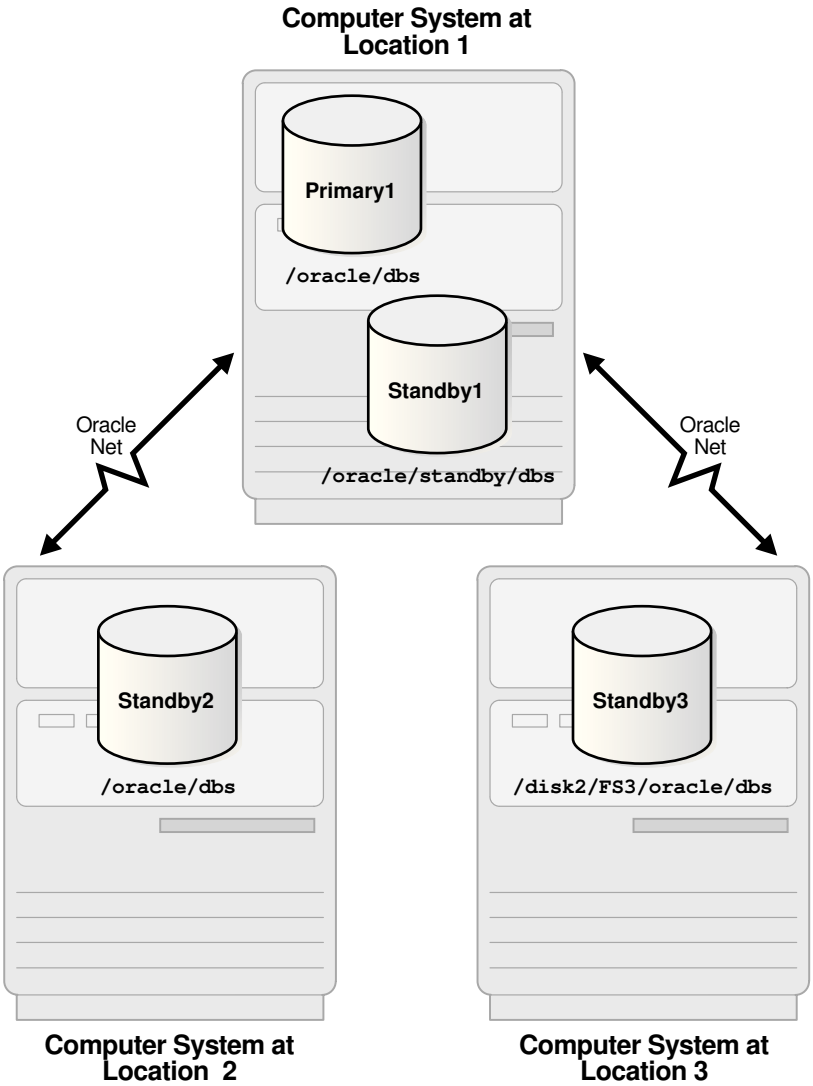


Table 2-1 describes possible configurations of primary and standby databases and the consequences of each.

Table 2-1 Standby Database Location and Directory Options

Standby System	Directory Structure	Consequences
Same as primary system	Different than primary system (required)	• 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 data files and archived redo log files and standby redo log files in the standby database control file. (See Set Primary Database Initialization Parameters.) • 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 data files (see Set Primary Database Initialization Parameters). • By locating the standby database on separate physical media, you safeguard the data on the primary database against disasters that destroy the primary system.

2.5 Moving the Location of Online Data Files

You can move the location of an online data file from one physical file to another physical file while the database is actively accessing the file.

To move the location of the file, you use the SQL statement `ALTER DATABASE MOVE DATAFILE`.

An operation performed with the `ALTER DATABASE MOVE DATAFILE` statement increases the availability of the database because it does not require that the database be shut down to move the location of an online data file.

You can perform an online move data file operation independently on the primary and on the standby (either physical or logical). The standby is not affected when a data file is moved on the primary, and vice versa.

On a physical standby, an online move data file operation can be executed while standby recovery is running if the instance that opens the database is in read-only mode. This functionality requires an Oracle Active Data Guard license.

2.5.1 Restrictions When Moving the Location of Online Data Files

There are restrictions when you move the location of online data files.

- You cannot use the SQL `ALTER DATABASE MOVE DATAFILE` command to rename or relocate an online data file on a physical standby that is a fast-start failover target if the standby is mounted, but not open.
- The online move data file operation cannot be executed on physical standby while standby recovery is running in a mounted but not open instance.
- The online move data file operation may get aborted if the standby recovery process takes the data file offline, shrinks the file, or drops the tablespace.
- On a primary database, the online move data file operation cannot be executed on a file that belongs to a pluggable database (PDB) that has been closed on all instances of the primary database.

See Also:

- *Oracle Database Administrator's Guide* for more information about renaming and relocating online data files
- *Oracle Database SQL Language Reference* for more information about the `ALTER DATABASE MOVE DATAFILE` statement

3
Creating a Physical Standby Database

You can manually create a physical standby database in maximum performance mode using asynchronous redo transport and real-time apply, the default Oracle Data Guard configuration.

See 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
- Using DBCA to Create a Data Guard Standby
- Creating a Physical Standby of a CDB
- Creating a PDB in a Primary Database

See Also:

- *Oracle Database Administrator's Guide* for information about creating and using server parameter files
- Enterprise Manager online help system for information about using the Oracle Data Guard broker graphical user interface (GUI) to automatically create a physical standby database
- Creating a Standby Database with Recovery Manager for information about alternative methods of creating a physical standby database that automate much of the process by using Oracle Recovery Manager (RMAN) and either backup based duplication or active duplication over a network
- *Oracle Data Guard Broker* for information about configuring a database so that it can be managed by Oracle Data Guard broker

Note:

If you are working in a multitenant container database (CDB) environment, then see Creating a Physical Standby of a CDB for information about behavioral differences from non-CDB environments. For instance, in a CDB environment, many DBA views have analogous CDB views that you should use instead.

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.

Perform the following tasks on the primary database to prepare for physical standby database creation:

- Enable Forced Logging
- Configure Redo Transport Authentication
- Configure the Primary Database to Receive Redo Data
- Set Primary Database Initialization Parameters
- 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

As part of preparing the primary database for standby database creation, you place the primary database in `FORCE LOGGING` mode.

You can do this after database creation using the following SQL statement:

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

When you issue this statement, the primary database must at least be mounted (and it can also be open). This statement can take a considerable amount of time to complete, because it waits for all unlogged direct write I/O to finish.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* for more information about the ramifications of specifying `FORCE LOGGING` mode

3.1.2 Configure Redo Transport Authentication

Oracle Data Guard uses Oracle Net sessions to transport redo data and control messages between the members of an Oracle 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`, and `FAL_SERVER` 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, then each member of an Oracle 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:

As of Oracle Database 12*c* Release 2 (12.2.0.1) password file changes done on a primary database are automatically propagated to standby databases. The only exception to this is far sync instances. Updated password files must still be manually copied to far sync instances because far sync instances receive redo, but do not apply it. Once the password file is up-to-date at the far sync instance, the redo is automatically propagated to any standby databases that are set up to receive redo logs from that far sync instance. The password file is updated on the standby when the redo is applied.

See Also:

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

3.1.3 Configure the Primary Database to Receive Redo Data

It is a best practice to configure the primary database to receive redo if this is the first time a standby database is added to the configuration.

The primary database can then quickly transition to the standby role and begin receiving redo data, if necessary.

To create a standby redo log, use the SQL `ALTER DATABASE ADD STANDBY LOGFILE` statement. For example:

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

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

See Configuring an Oracle Database to Receive Redo Data for a discussion of how to determine the size of each log file and the number of log groups, as well as other background information about managing standby redo logs.

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.

The following example shows the primary role initialization parameters that you maintain on the primary database. This example represents an Oracle Data Guard configuration with a primary database located in Chicago and one physical standby database located in Boston. The parameters shown in this example 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

```
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=USE_DB_RECOVERY_FILE_DEST
  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'
REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE
LOG_ARCHIVE_FORMAT=%t_%s_%r.arc
```

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 Configure the Primary Database to Receive Redo Data).

The following 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.

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

Specifying the initialization parameters shown above sets up the primary database to resolve gaps, converts new data file 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 the previous two examples.

Parameter	Recommended Setting
DB_NAME	On a primary database, specify the name used when the database was created. On a physical standby database, use the DB_NAME of the primary database.
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	The DG_CONFIG attribute of this parameter must be explicitly set on each database in an Oracle Data Guard configuration to enable full Oracle Data Guard functionality. Set DG_CONFIG to a text string that contains the DB_UNIQUE_NAME of each database in the configuration, with each name in this list separated by a comma.
CONTROL_FILES	Specify the path name for the control files on the primary database. 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. • 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 fast 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, Oracle Data Guard automatically uses the LOG_ARCHIVE_DEST_1 initialization parameter (if it has not already been set) as the default destination for local archiving. Also, see LOG_ARCHIVE_DEST_n Parameter Attributes for complete LOG_ARCHIVE_DEST_*n* information.
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).
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.
DB_FILE_NAME_CONVERT	Specify the path name and filename location of the standby database data files followed by the primary location. This parameter converts the path names of the primary database data files to the standby data file path names. 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 standby database online redo log files followed by the primary location. This parameter converts the path names of the primary database log files to the path names on the standby database. Multiple pairs of paths may be specified by this parameter.

Parameter	Recommended Setting
STANDBY_FILE_MANAGEMENT	Set to AUTO so when data files are added to or dropped from the primary database, corresponding changes are made automatically to the standby database.

> **Note:**
>
> 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, then you must put the primary database in ARCHIVELOG mode and enable automatic archiving.

Issue the following SQL statements:

```
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

These are the tasks you perform to create a physical standby database.

The information in this topic is written at a level of detail that requires you to already have a thorough understanding of the following topics:

- Database administrator authentication
- Database initialization parameters
- Managing redo logs, data files, and control files
- Managing archived redo logs
- Fast recovery areas
- Oracle Net configuration

Table 3-1 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.

Table 3-1 Creating a Physical Standby Database

Task	Database
Create a Backup Copy of the Primary Database Data Files	Primary
Create a Control File for the Standby Database	Primary
Create a Parameter File for the Standby Database	Primary
Copy Files from the Primary System to the Standby System	Primary
Set Up the Environment to Support the Standby Database	Standby
Start the Physical Standby Database	Standby
Verify the Physical Standby Database Is Performing Properly	Standby

3.2.1 Creating a Physical Standby Task 1: Create a Backup Copy of the Primary Database Data Files

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.

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 Creating a Physical Standby Task 2: Create a Control File for the Standby Database

Create the control file for the standby database (the primary database does not have to be open, but it must at least be mounted).

The following is an example of creating the control file for the standby database:

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

The `ALTER DATABASE` command designates the database that is to operate in the standby role; in this case, a database named `boston`.

You cannot use a single control file for both the primary and standby databases. They must each have their own file.

> **Note:**
>
> If a control file backup is taken on the primary and restored on a standby (or vice versa), then the location of the snapshot control file on the restored system is configured to be the default. (The default value for the snapshot control file name is platform-specific and dependent on Oracle home.) Manually reconfigure it to the correct value using the RMAN `CONFIGURE SNAPSHOT CONTROLFILE` command.

3.2.3 Creating a Physical Standby Task 3: Create a Parameter File for the Standby Database

Create a parameter file (PFILE) from the server parameter file (SPFILE) used by the primary database.

Perform the following steps:

1. Issue a SQL statement such as the following:

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

   In Set Up the Environment to Support the Standby Database, you then create a server parameter file from this parameter file, after it has been modified to contain parameter values appropriate for use at the physical standby database.

2. Modify the parameter values in the parameter file created in the previous step.

   Although most of the initialization parameter settings in the parameter file are also appropriate for the physical standby database, some modifications must be made.

   Example 3-1 shows, in bold typeface, the parameters created earlier on the primary that must be changed.

Example 3-1 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='/chicago/','/boston/'
LOG_ARCHIVE_FORMAT=log%t_%s_%r.arc
LOG_ARCHIVE_DEST_1=
 'LOCATION=USE_DB_RECOVERY_FILE_DEST
  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'
REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE
STANDBY_FILE_MANAGEMENT=AUTO
FAL_SERVER=chicago
```

.
.
.

Ensure the COMPATIBLE initialization parameter is set to the same value on both the primary and standby databases. If the values differ, then redo transport services may be unable to transmit redo data from the primary database to the standby databases.

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 3-1 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 3-1 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.
DB_FILE_NAME_CONVERT	Specify the path name and filename location of the primary database data files followed by the standby location. This parameter converts the path names of the primary database data files to the standby data file path names.
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.
LOG_ARCHIVE_DEST_*n*	Specify where the redo data is to be archived. In Example 3-1: • 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 transmits redo data to the remote Chicago destination. **Note:** If a fast 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, then Oracle Data Guard automatically uses the LOG_ARCHIVE_DEST_1 initialization parameter (if it has not already been set) as the default destination for local archiving. Also, see LOG_ARCHIVE_DEST_n Parameter Attributes 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.

> **Note:**
>
> 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 Creating a Physical Standby Task 4: Copy Files from the Primary System to the Standby System

Ensure that all required directories are created. Use an operating system copy utility to copy binary files from the primary system to their correct locations on the standby system.

The binary files to copy are as follows:

- Database backup created in Create a Backup Copy of the Primary Database Data Files
- Standby control file created in Create a Control File for the Standby Database
- Initialization parameter file created in Create a Parameter File for the Standby Database

3.2.5 Creating a Physical Standby Task 5: Set Up the Environment to Support the Standby Database

Set up the environment by creating a Windows-based service, a password file and an SPFILE, and setting up the Oracle Net environment.

Perform the following steps:

1. If the standby database is going to be hosted on a Windows system, then use the ORADIM utility to create a Windows service. For example:

   ```
   oradim –NEW –SID boston –STARTMODE manual
   ```

   The ORADIM utility automatically determines the username for which this service should be created and prompts for a password for that username (if that username needs a password). See *Oracle Database Platform Guide for Microsoft Windows* for more information about using the ORADIM utility.

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

   This step is optional if operating system authentication is used for administrative users and if SSL is used for redo transport authentication. If not, then copy the remote login password file from the primary database to the appropriate directory on the physical standby database system.

   Any subsequent changes to the password file on the primary are automatically propagated to the standby. Changes to a password file can include when administrative privileges (`SYSDG`, `SYSOPER`, `SYSDBA`, and so on) are granted or revoked, and when passwords of any user with administrative privileges is

changed. Updated password files must still be manually copied to far sync instances because far sync instances receive redo, but do not apply it. Once the password file is up-to-date at the far sync instance, the redo containing the password update at the primary is automatically propagated to any standby databases that are set up to receive redo from that far sync instance. The password file is updated on the standby when the redo is applied.

3. Configure and start a listener on the standby system if one is not already configured.

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

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 are to be used by redo transport services. The Net service names in this example are `chicago` and `boston`.

   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* for more information about service names.

5. 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 Task 3. For example:

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

6. If the primary database has a database encryption wallet, then 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.

3.2.6 Creating a Physical Standby Task 6: Start the Physical Standby Database

These are the steps to start the physical standby database and Redo Apply.

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

   ```
   SQL> STARTUP MOUNT;
   ```

2. Restore the backup of the data files taken in Create a Backup Copy of the Primary Database Data Files and copied in Copy Files from the Primary System to the Standby System on the standby system.
3. On the standby database, issue the following command to start Redo Apply:

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

   The statement includes the `DISCONNECT FROM SESSION` option so that Redo Apply runs in a background session.

3.2.7 Creating a Physical Standby Task 7: Verify the Physical Standby Database Is Performing Properly

After 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.

On the standby database, query the `V$MANAGED_STANDBY` view to verify that redo is being transmitted from the primary database and applied to the standby database.

The following is an example of querying the `V$MANAGED_STANDBY` view:

```
SQL> SELECT CLIENT_PROCESS, PROCESS, THREAD#, SEQUENCE#, STATUS FROM
V$MANAGED_STANDBY WHERE CLIENT_PROCESS='LGWR' OR PROCESS='MRP0';

CLIENT_PROCESS PROCESS   THREAD#    SEQUENCE#  STATUS
-------------- --------- ---------- ---------- ------------
N/A            MRP0      1          80         APPLYING_LOG
LGWR           RFS       1          80         IDLE
```

The query output should show one line for the primary database with a `CLIENT_PROCESS` of `LGWR`. This indicates that redo transport is functioning correctly and the primary redo thread is being sent to the standby.

> **Note:**
>
> If the Primary database is an Oracle RAC database, then the output contains one line with a `CLIENT_PROCESS` of `LGWR` for each primary instance that is currently active.

The query output should also show one line for the MRP. If the MRP status shows `APPLYING_LOG` and the `SEQUENCE#` is equal to the sequence number currently being sent by the primary database, then the standby has resolved all gaps and is currently in real-time apply mode.

> **Note:**
>
> The MRP may show a `SEQUENCE#` older than the sequence number currently being sent from the primary. This indicates that it is applying archive log files that were sent as a gap and it has not yet caught up. Once all gaps are resolved, the same query shows that the MRP is applying the current `SEQUENCE#`.

3.3 Creating a Physical Standby: Post-Creation Steps

After the physical standby database is running, you can upgrade the data protection mode, and you can enable Flashback Database.

- Upgrade the data protection mode

  Oracle Data Guard Protection Modes provides information about configuring the different data protection modes.

- 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 data files from backup nor the extensive application of redo data. You can enable Flashback Database on the primary database, the standby database, or both.

> **See Also:**
>
> - Converting a Failed Primary Into a Standby Database Using Flashback Database and Using Flashback Database After Issuing an Open Resetlogs Statement for scenarios showing how to use Flashback Database in an Oracle Data Guard environment
> - *Oracle Database Backup and Recovery User's Guide* for more information about Flashback Database

3.4 Using DBCA to Create a Data Guard Standby

The Database Configuration Assistant (DBCA) can also be used as a simple command-line method to create an Oracle Data Guard physical standby database.

The DBCA command qualifier used to create the physical standby database is `createDuplicateDB` .
DBCA can only be used to create standby databases for non-multitenant primary databases. In addition, this capability creates only single instance standby databases, not Oracle Real Application Clusters (Oracle RAC) databases. If required, the standby can then be converted to an Oracle RAC standby database, either manually or using Oracle Enterprise Manager Cloud Control.

The basic `createDuplicateDB` command has the following syntax:

```
dbca -createDuplicateDB
    -gdbName global_database_name
    -primaryDBConnectionString easy_connect_string_to_primary
    -sid database_system_identifier
    [-createAsStandby
        [-dbUniqueName db_unique_name_for_standby]]
    [-customScripts scripts_list]
```

For more information about `createDuplicateDB` options, including the use of custom scripts, see *Oracle Database Administrator's Guide*.

In the following two examples the primary database is `chicago` and it resides on the primary system `myprimary.domain`. Each example creates a physical standby on the system on which the command is executed, `boston`. The `initParams` parameter is used in the examples to show how other DBCA parameters can be used in the standby creation command. In these examples, `initParams` is used to explicitly set the `INSTANCE_NAME` of the standby to match the `DB_UNIQUE_NAME`, `boston`.

This first example creates the standby database without any custom scripts being executed afterward.

```
dbca –silent -createDuplicateDB -primaryDBConnectionString  myprimary.domain:1523/
chicago.domain
-gdbName chicago.domain -sid boston -initParams instance_name=boston –createAsStandby

Enter SYS user password:
Listener config step
33% complete
Auxiliary instance creation
66% complete
RMAN duplicate
100% complete
Look at the log file " /u01/app/oracle/product/12.2.0/dbhome_1/cfgtoollogs/dbca/
chicago/chicago.log" for further details.
```

The following example is exactly the same as the previous example, except that it runs a SQL script named `/tmp/test.sql` which can be used to perform post-creation operations.

```
dbca -silent -createDuplicateDB -primaryDBConnectionString  myprimary.domain:1523/
chicago.domain
-gdbName chicago.domain -sid boston -initParams instance_name=boston -
createAsStandby  -customScripts /tmp/test.sql

Enter SYS user password:

Listener config step
25% complete
Auxiliary instance creation
50% complete
RMAN duplicate
75% complete
Running Custom Scripts
100% complete
Look at the log file " /u01/app/oracle/product/12.2.0/dbhome_1/cfgtoollogs/dbca/
chicago/chicago.log" for further details.
```

> **Note:**
>
> Even though it is required to have a listener running on the physical standby system, it is not necessary to configure the Oracle Net service names for the databases on either system to execute these commands. In these examples, the Easy Connect naming method was used to create a connection to the primary database, `Chicago`, to complete creation of the standby, `Boston` . Before adding the new standby to the Data Guard configuration you would first configure Oracle Net service name descriptors on both systems, as described in Step 4 in Creating a Physical Standby Task 5: Set Up the Environment to Support the Standby Database.

When these commands complete without any errors, the physical standby `Boston` is ready to be added to your Data Guard configuration. As part of adding it, you would need to define the Data Guard parameters in `Chicago` and `Boston` as shown in Creating a Physical Standby Task 3: Create a Parameter File for the Standby Database. Optionally, if you have an Oracle Data Guard broker configuration, you could use the broker `ADD DATABASE` command to add the new standby to your configuration (see *Oracle Data Guard Broker*).

3.5 Creating a Physical Standby of a CDB

You can create a physical standby of a multitenant container database (CDB) just as you can create a physical standby of a regular primary database.

The following are some of the behavioral differences to be aware of when you create and use a physical standby of a CDB:

- If you execute a switchover or failover operation, the entire CDB undergoes the role change. If you used the `ENABLED_PDBS_ON_STANDBY` intialization parameter, then be aware of the possibility that not every PDB is present in both the primary and the standby databases.
- The database role is defined at the CDB level, not at the individual container level.
- Any DDL related to role changes must be executed in the root container because a role is associated with an entire CDB. Individual pluggable databases (PDBs) do not have their own roles.
- In a physical standby of a CDB, the syntax of SQL statements is generally the same as for noncontainer databases. However, the effect of some statements, including the following, may be different:
  - `ALTER DATABASE RECOVER MANAGED STANDBY` functions only in the root container; it is not allowed in a PDB.
  - A role is associated with an entire CDB; individual PDBs do not have their own roles. Therefore, the following role change DDL associated with physical standbys affect the entire CDB:

    ```
    ALTER DATABASE SWITCHOVER TO target_db_name

    ALTER DATABASE ACTIVATE PHYSICAL STANDBY
    ```

- The `ALTER PLUGGABLE DATABASE [OPEN|CLOSE]` SQL statement is supported on the standby, provided you have already opened the root container.

- The `ALTER PLUGGABLE DATABASE RECOVER` statement is not supported on the standby. (Standby recovery is always at the CDB level.)
- To administer a multitenant environment, you must have the `CDB_DBA` role.
- Oracle recommends that the standby database have its own keystore.
- In a multitenant environment, the redo must be shipped to the root container of the standby database.

  The following is an example of how to determine whether redo is being shipped to the root container. Suppose your primary database has the following settings:

  ```
  LOG_ARCHIVE_DEST_2='SERVICE=boston ASYNC VALID_FOR=(ONLINE_LOGFILES,
  PRIMARY_ROLE) DB_UNIQUE_NAME=boston'
  ```

  Redo is being shipped to `boston`. The container ID (`CON_ID`) for the root container is always 1, so you must make sure that the `CON_ID` is 1 for the service `boston`. To do this, check the service name in the `tnsnames.ora` file. For example:

  ```
  boston = (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=boston-server)(PORT=1521))
  (CONNECT_DATA=(SERVICE_NAME=boston.us.example.com))
  ```

  The service name for `boston` is `boston.us.example.com`.

  On the standby database, query the `CDB_SERVICES` view to determine the `CON_ID`. For example:

  ```
  SQL> SELECT NAME, CON_ID FROM CDB_SERVICES;

  NAME                                 CON_ID
  ---------------------------------------------
  boston.us.example.com                 1
  ```

  The query result shows that the `CON_ID` for `boston` is 1.

> **See Also:**
>
> - *Oracle Database Concepts* for more information about CDBs
> - *Oracle Database Advanced Security Guide* for more information about creating keystores

3.6 Creating a PDB in a Primary Database

In an Oracle Data Guard configuration, a PDB on a primary database is created in the same way that a PDB on a regular database is created.

The steps to create a PDB on a regular database are documented in the *Oracle Database Administrator's Guide*. Before following those steps, note the following:

- In Oracle Database 12*c* Release 1 (12.1), you could only specify whether a PDB was created and recovered in all (`ALL`) standbys or in no (`NONE`) standbys when adding a PDB to the primary database. As of Oracle Database 12*c* Release 2 (12.2.0.1), you can specify a subset of PDBs to be replicated on a physical standby of a multitenant container database (CDB), instead of having to choose either all PDBs or none. To do so, use the `ENABLED_PDBS_ON_STANDBY` initialization

parameter to specify a list of PDBs or use the enhanced STANDBYS qualifier on the CREATE PLUGGABLE DATABASE statement, or both. PDBs that are not enabled on a standby CDB can remain disabled (true SUBSET Standby) or they can be enabled at a later date when all the required files are available at the standby CDB.

- The ENABLED_PDBS_ON_STANDBY parameter is valid only on a physical standby; it is ignored by primary databases. (It can be set on a primary database to be used if that database ever becomes a standby database.) It can be used to specify which PDBs should or should not be enabled on a physical standby database. If the parameter is not specified, then all PDBs in the CDB are created on the standby unless the STANDBYS clause is used. See ENABLED_PDBS_ON_STANDBY for more information about this parameter.
- To specify in which standby CDBs the new PDB being created is to be included, you can also use the STANDBYS clause of the SQL CREATE PLUGGABLE DATABASE statement. The syntax is as follows:

```
create pluggable database … STANDBYS={('cdb_name', 'cdb_name', ...) | NONE | ALL
[EXCEPT ('cdb_name', 'cdb_name', ...)]}
```

  - cdb_name is the DB_UNIQUE_NAME for the physical standbys in which the PDB is to be included
  - NONE excludes the PDB being created from all standby CDBs. When a PDB is excluded from all standby CDBs, the PDB's data files are offline and marked as unnamed on all of the standby CDBs. Any new standby CDBs that are instantiated after the PDB has been created must disable the PDB for recovery explicitly to exclude it from the standby CDB. It is possible to enable a PDB on a standby CDB after it was excluded on that standby CDB.
  - ALL (the default) includes the PDB being created in all standby CDBs.
  - EXCEPT cdb_name includes the PDB being created in all standby CDBs except for those CDBs listed in this clause by their DB_UNIQUE_NAME.

  Parentheses are required around the list of CDB names and each name must be enclosed within single quotation marks. The value of DB_UNIQUE_NAME can be up to 30 characters and is case insensitive. The following characters are valid in a database name: alphanumeric characters, underscore (_), number sign (#), and dollar sign ($).

> **Note:**
>
> The EXCEPT clause is available starting with Oracle Database 12*c* Release 2 (12.2.0.1).

- To create a PDB as a local clone from a different PDB or from the seed PDB within the same primary CDB, copy the data files that belong to the source PDB over to the standby database. (This step is not necessary in an Oracle Active Data Guard environment because the data files are copied automatically at the standby when the PDB is created on the standby database.)

  To perform a remote clone of a PDB from another CDB into the primary CDB, you must use the STANDBY=NONE clause and then copy the files and enable recovery by following the steps in the My Oracle Support note 2049127.1 at http://support.oracle.com.

- To create a PDB from an XML file, copy the data files specified in the XML file to the standby database.

  If your standby database has the Oracle Active Data Guard option enabled (open read-only), then copy to it the same set of PDB data files that are to be plugged into the primary database. To minimize disruptions to managed standby recovery or database sessions running on systems that have Oracle Active Data Guard enabled, you must copy these files to the standby database before plugging in the PDB at the primary database. Ensure that the files are copied to an appropriate location where they can be found by managed standby recovery:

  – If data files reside in standard operating system file systems, then the location of the files at the standby database are based on the value of the `DB_FILE_NAME_CONVERT` parameter. For more details about setting primary database initialization parameters, see Set Primary Database Initialization Parameters

  – If data files reside in ASM, then use the ASMCMD utility to copy the files to the following location at the standby database:

    ```
    <db_create_file_dest>/<db_unique_name>/<GUID>/datafile
    ```

    The `GUID` parameter is the global unique identifier assigned to the PDB; once assigned, it does not change. To find the value of the `GUID` parameter, query the `V$CONTAINERS` view before unplugging the PDB from its original source container. The following example shows how to find the value of the `GUID` parameter for the PDB whose PDB container ID in the source container is `3`:

    ```
    SELECT guid
      FROM V$CONTAINERS
     WHERE con_id=3;

    GUID

    D98C12257A951FC4E043B623F00A7AF5
    ```

    In this example, if the value of the `DB_CREATE_FILE_DEST` parameter is `+DATAFILE` and the value of the `DB_UNIQUE_NAME` parameter is `BOSTON`, then the data files are copied to:

    ```
    +DATAFILE/BOSTON/D98C12257A951FC4E043B623F00A7AF5/datafile
    ```

The path name of the data files on the standby database must be the same as the resultant path name when you create the PDB on the primary, unless the `DB_FILE_NAME_CONVERT` database initialization parameter has been configured on the standby. In that case, the path name of the data files on the standby database is the path name on the primary with `DB_FILE_NAME_CONVERT` applied.

> **See Also:**
>
> - *Oracle Database SQL Language Reference* for more information about the SQL statement `CREATE PLUGGABLE DATABASE`

4
Creating a Logical Standby Database

There are a number of steps involved in creating a logical standby database, including prerequisites and post-creation tasks.

See the following topics for information about creating a logical standby database.

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

> **See Also:**
>
> - *Oracle Database Administrator's Guide* for information about creating and using server parameter files
> - Oracle Enterprise Manager Cloud Control online help system for information about using the Oracle Data Guard broker graphical user interface (GUI) to automatically create a logical standby database.

> **Note:**
>
> If you are working in a multitenant container database (CDB) environment, then see Creating a Logical Standby of a CDB for information about behavioral differences from non-CDB environments. For instance, in a CDB environment, many DBA views have analogous CDB views that you should use instead. For example, the view `CDB_LOGSTDBY_NOT_UNIQUE` contains the same data as shown in `DBA_LOGSTDBY_NOT_UNIQUE` view, but it has an additional column indicating the PDB name.

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.

Perform the following tasks on the primary database to prepare for logical standby database creation:

- Determine Support for Data Types and Storage Attributes for Tables
- Ensure Table Rows in the Primary Database Can Be Uniquely Identified

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 Unsupported Tables for information about specific SQL queries you can use to determine if there are any unsupported data types or storage attributes.

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

The ROWIDs contained in the redo records generated by the primary database cannot be used to identify the corresponding row in the logical standby database.

This is because 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.

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.
- If there is no primary key and no nonnull unique constraint/index, then all columns with a declared maximum length of 4000 bytes are logged as part of the UPDATE statement to help identify the modified row. There are some requirements and restrictions with respect to supported data types. See the following sections for more information:
  - Supported Table Storage Types
  - Unsupported Table Storage Types
- 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.

1. 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';
   ```

   This query may take a few minutes to run.

2. If your application ensures the rows in a table are unique, then 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 assumes 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 uniquely identify each row in the table. If such uniqueness is not present, then SQL Apply does 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
- Create a Primary Key RELY Constraint 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

These are the tasks you perform to create a logical standby database.

Table 4-1 Creating a Logical Standby Database

Task	Database
Create a Physical Standby Database	Primary
Stop Redo Apply on the Physical Standby Database	Standby
Prepare the Primary Database to Support a Logical Standby Database	Primary
Transition to a Logical Standby Database	Standby
Open the Logical Standby Database	Standby
Verify the Logical Standby Database Is Performing Properly	Standby

4.2.1 Creating a Logical Standby Task 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 Creating a Physical Standby Database to create a physical standby database.

4.2.2 Creating a Logical Standby Task 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 Build a Dictionary in the Redo Data).

To stop Redo Apply, issue the following statement on the physical standby database:

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

4.2.3 Creating a Logical Standby Task 3: Prepare the Primary Database to Support a Logical Standby Database

As part of creating a logical standby database, you must prepare the primary database to support a logical standby.

See 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

You may have to modify certain parameters when you prepare to switch the role of a primary database.

In Set Primary Database Initialization Parameters, 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.

The following table describes the archival processing defined by the changed initialization parameters shown in Example 4-1.

LOG_ARCHIVE_DEST_n	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/`.

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 parameters, 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.

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 affect the timeliness of this command.

> **Note:**
>
> In databases created using Oracle Database 11*g* Release 2 (11.2) or later, supplemental logging information is automatically propagated to any existing physical standby databases. However, for databases in earlier releases, or if the database was created using an earlier release and then upgraded to 11.2, you must check whether supplemental logging is enabled at the physical standby(s) if it is also enabled at the primary database. If it is not enabled at the physical standby(s), then before performing a switchover or failover, you must enable supplemental logging on all existing physical standby databases. To do so, issue the following SQL statement on each physical standby:
>
> ```
> SQL>  ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY, UNIQUE INDEX)
> COLUMNS;
> ```
>
> If you do not do this, then any logical standby that is also in the same Oracle Data Guard configuration is unusable if a switchover or failover is performed to one of the physical standby databases. If a switchover or failover has already occurred and supplemental logging was not enabled, then you must recreate all logical standby databases.

> **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 Creating a Logical Standby Task 4: Transition to a Logical Standby Database

There are some necessary tasks you must perform to prepare the physical standby database to transition to a logical standby database.

The following topics describe these tasks:

- 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, then 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 that is different from the primary database 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 updates 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, then 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.
>
> 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 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, then this command never completes. You can cancel the SQL statement by issuing the `ALTER DATABASE RECOVER MANAGED STANDBY DATABASE CANCEL` statement from another SQL session.

> **Caution:**
>
> In releases prior to Oracle Database 11*g*, 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 causes 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, then 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.

> **Note:**
>
> If database compatibility is set to 11.1 or later, you can use the fast recovery area to store remote archived logs. To do this, you need to set only the following parameters (assuming you have already set the DB_RECOVERY_FILE_DEST and DB_RECOVERY_FILE_DEST_SIZE parameters):
>
> ```
> LOG_ARCHIVE_DEST_1=
>   'LOCATION=USE_DB_RECOVERY_FILE_DEST
>    DB_UNIQUE_NAME=boston'
> ```
>
> Because you are using the fast recovery area, it is not necessary to specify the VALID_FOR parameter. Its default value is (ALL_LOGFILES,ALL_ROLES) and that is the desired behavior in this case. LOG_ARCHIVE_DEST_1 is used for all log files, both online (primary) and standby.

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

LOG_ARCHIVE_DEST_n	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, register a skip handler and provide SQL Apply with a replacement DDL string to execute by converting the path names of the primary database data files to the standby data file path names. See the `DBMS_LOGSTDBY` package for information about the `SKIP` procedure.

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
```

4.2.5 Creating a Logical Standby Task 5: Open the Logical Standby Database

Use an `ALTER DATABASE` SQL statement to open the newly created logical standby.

For example, issue the following statement (do not supply the `RESETLOGS` option if the logical standby was created using the `KEEP IDENTITY` option):

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

> **Note:**
>
> If you started with an Oracle RAC physical standby database, then 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, then 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 data files that belong to the primary database could possibly be damaged if SQL Apply attempted to re-execute 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. (This is not true if the logical standby was created using the `KEEP IDENTITY` option.)

Note:

If you are creating the logical standby database to perform a rolling upgrade of the Oracle Database software, and you are concerned about updates to objects that may not be supported by SQL Apply, then Oracle recommends that you use the `DBMS_LOGSTDBY` PL/SQL procedure. At the 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');
```

This captures information about any transactions running on the primary that are not supported by logical standby. When the upgrade is complete and before you switch production to the new version, check this table. If nothing is recorded, then you know everything was replicated. If something is recorded, then you can choose to either take corrective action or abandon the upgrade.

See Also:

- Customizing Logging of Events in the DBA_LOGSTDBY_EVENTS View 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

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

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

4.2.6 Creating a Logical Standby Task 6: Verify the Logical Standby Database Is Performing Properly

After you create a logical standby database, it is important to verify that it is performing properly.

See the following topics:

- Redo Transport Services
- Managing a Logical Standby Database

4.3 Creating a Logical Standby: Post-Creation Steps

> **Note:**
>
> The conversion of the physical standby database to a logical standby database happens in two phases:
>
> 1. As part of the `ALTER DATABASE RECOVER TO LOGICAL STANDBY` statement (unless you have specified the `KEEP IDENTITY` clause), the DBID of the database is changed.
> 2. As part of the first successful invocation of `ALTER DATABASE START LOGICAL STANDBY APPLY` statement, the control file is updated to make it consistent with that of the newly created logical standby database.
>
>   After you have successfully invoked the `ALTER DATABASE START LOGICAL STANDBY APPLY` statement, take a full backup of the logical standby database, because the backups taken from the primary database cannot be used to restore the logical standby database.

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 Oracle 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 data files from backup nor the extensive application of redo data. You can enable Flashback Database on the primary database, the standby database, or both.

See Also:

- Converting a Failed Primary Into a Standby Database Using Flashback Database and Using Flashback Database After Issuing an Open Resetlogs Statement for scenarios showing how to use Flashback Database in an Oracle Data Guard environment.
- *Oracle Database Backup and Recovery User's Guide* for more information about Flashback Database

4.4 Creating a Logical Standby of a CDB

You can create a logical standby of a multitenant container database (CDB) just as you can create a logical standby of a regular primary database.

The following are some of the behavioral differences to be aware of when you create and use a logical standby of a CDB:

- The database role is defined at the CDB level, not at the pluggable database (PDB) container level.
- If you execute a switchover or failover operation, then the entire CDB undergoes the role change.
- Any DDL related to role changes must be executed while connected to the root container of the CDB.
- As with a regular logical standby, a logical standby of a CDB operates a single pool of processes that mine the redo stream once, but the responsibility is shared for updating all of the PDBs and the root container of the CDB.
- You are not required to have the same set of PDBs at the primary and standby. However, only tables that exist in the same container at both the primary and standby are replicated.
- In general, logical standby PL/SQL interfaces which modify global configuration attributes, such as `DBMS_LOGSTDBY.APPLY_SET`, are executed in the root container. However, `DBMS_LOGSTDBY.INSTANTIATE_TABLE` must be called inside the container where the table of interest resides, and the `DBMS_LOGSTDBY.SKIP` procedure must be called inside the container of interest.
- Logical standby views are enhanced to provide container names where appropriate. Many DBA views have analogous CDB views whose names begin with CDB. For example, the view `CDB_LOGSTDBY_NOT_UNIQUE` contains the same data as shown in `DBA_LOGSTDBY_NOT_UNIQUE` view, but it has an additional column indicating the PDB name. When the `CDB_LOGSTDBY_NOT_UNIQUE` view is queried in the root it shows data for all databases in the CDB.
- In a logical standby of a CDB, the syntax of SQL statements is generally the same as for noncontainer databases. However, the effect of some statements, including the following, may be different:
  - `ALTER DATABASE RECOVER TO LOGICAL STANDBY` functions only in the CDB; it is not allowed in a PDB.

- A role is associated with an entire CDB; individual PDBs do not have their own roles. Therefore, the following role change DDL associated with logical standbys affect the entire CDB:

  ```
  ALTER DATABASE [PREPARE|COMMIT] TO SWITCHOVER
  ```

  ```
  ALTER DATABASE ACTIVATE LOGICAL STANDBY
  ```

- `ALTER DATABASE [START|STOP] LOGICAL STANDBY APPLY` functions only in the root container and affects the entire CDB. This statement is not allowed on a PDB.
- `ALTER DATABASE GUARD` functions only in the root container and affects the entire CDB. For example, if an `ALTER DATABASE GUARD ALL` statement is issued, then user activity in the root and in all PDBs is restricted.
- To administer a multitenant environment, you must have the `CDB_DBA` role.

See Also:

- *Oracle Database Concepts* for more information about CDBs
- *Oracle Database PL/SQL Packages and Types Reference* for more information about using the `DBMS_LOGSTDBY.SKIP` procedure in containers

5

Using Far Sync Instances

An Oracle Data Guard far sync instance is a remote Oracle Data Guard destination that accepts redo from the primary database and then ships that redo to other members of the Oracle Data Guard configuration.

A far sync instance manages a control file, receives redo into standby redo logs (SRLs), and archives those SRLs to local archived redo logs, but that is where the similarity with standbys ends. A far sync instance does not have user data files, cannot be opened for access, cannot run redo apply, and can never function in the primary role or be converted to any type of standby database.

Far sync instances are part of the Oracle Active Data Guard Far Sync feature, which requires an Oracle Active Data Guard license.

A far sync instance consumes very little disk and processing resources, yet provides the ability to failover to a terminal destination with zero data loss, as well as offload the primary database of other types of overhead (for example, redo transport).

All redo transport options available to a primary when servicing a typical standby destination are also available to it when servicing a far sync instance. And all redo transport options are available to a far sync instance when servicing terminal destinations (for example, performing redo transport compression, if you have a license for the Oracle Advanced Compression option).

Many configurations have a primary database shipping redo to a standby database using asynchronous transport at the risk of some data loss at failover time. Using synchronous redo transport to achieve zero data loss may not be a viable option because of the impact on the commit response times at the primary due to network latency between the two databases.

Creating a far sync instance close to the primary has the benefit of minimizing impact on commit response times to an acceptable threshold (due to the smaller network latency between primary and far sync instance) while allowing for higher data protection guarantees -- if the primary were to fail, and assuming the far sync instance was synchronized at the time of the failure, the far sync instance and the terminal standby would coordinate a final redo shipment from the far sync instance to the standby to ship any redo not yet available to the standby and then perform a zero-data-loss failover.

See the following topics:

5.1 Creating a Far Sync Instance

Creating a far sync instance is similar to creating a physical standby except that data files do not exist at the far sync instance.

Therefore, on a far sync instance there is no need to copy data files or restore data files from a backup. Once the far sync instance has been created, the configuration is modified to send redo synchronously from the primary database to the far sync instance in Maximum Availability mode and the far sync instance then forwards the redo asynchronously in real time. Lastly, the original asynchronous standby (referred to as the terminal standby) is configured to act as the alternate to the far sync instance in the event that communication with the far sync instance is interrupted.

> **Note:**
>
> In a configuration that contains a far sync instance, there must still be a direct network connection between the primary database and the remote standby database. The direct connection between the primary and the remote standby is used to perform health checks and switchover processing tasks. It is not used for redo transport unless the standby has been configured as an alternate destination in case the far sync instance fails and there is no alternate far sync configured to maintain the protection level.

5.1.1 Creating and Configuring a Far Sync Instance

Take the following steps to create a far sync instance:

1. Create the control file for the far sync instance, as shown in the following example (the primary database does not have to be open, but it must at least be mounted):

   ```
   SQL> ALTER DATABASE CREATE FAR SYNC INSTANCE CONTROLFILE AS -
   > '/arch2/chicagoFS/control01.ctl';
   ```

   The resulting control file enables `chicagoFS` to operate as a far sync instance that receives redo from primary database `chicago`. The path and file name shown are just an example; you could use any path or file name that you want.

2. Create a parameter file (PFILE) from the server parameter file (SPFILE) used by the primary database. Although most of the initialization settings in the parameter file are also appropriate for the far sync instance, some modifications must be made. For example, on a far sync instance, the `DB_FILE_NAME_CONVERT` and `LOG_FILE_NAME_CONVERT` parameters must be set, and the `DB_UNIQUE_NAME` of the far sync instance and the location of the far sync instance control file must be modified. Example 5-1 shows sample parameter file content for a far sync instance with a `DB_UNIQUE_NAME` of `chicagoFS`.

3. Create a server parameter file (SPFILE) from the edited parameter file (PFILE) to facilitate any subsequent changes to parameter values. If you do not use an SPFILE, then a warning is returned in the `SHOW CONFIGURATION` output when the far sync instance is added to an Oracle Data Guard broker configuration.

4. Use an operating system copy utility to copy the far sync instance control file created in Step 1 and the server parameter file (SPFILE) created in Step 3 from the primary system to the appropriate locations on the far sync instance system.
5. Create standby redo logs in the same way they are created for a regular standby. See Managing Standby Redo Logs.

   Because the `LOG_FILE_NAME_CONVERT` parameter was specified on the far sync instance (see Example 5-1), the standby redo logs are created automatically when redo transport begins from the primary, if they were created on the primary as described in Configure the Primary Database to Receive Redo Data.

> **Note:**
>
> Standby redo log files used at the far sync instance cannot be shared with other databases. Therefore, all relevant considerations discussed in Standby Database Directory Structure Considerations for standby redo log files also apply at the far sync instance.

6. If the far sync instance is to be hosted on a Windows system, use the ORADIM utility to create a Windows service. For example:

   ```
   oradim -NEW -SID ChicagoFS -STARTMODE manual
   ```

   After the far sync instance is created and running you can change the `STARTMODE` to `auto` to enable automatic startup of the far sync instance

   The ORADIM utility automatically determines the username for which this service should be created and prompts for a password for that username (if that username needs a password). See *Oracle Database Platform Guide for Microsoft Windows* for more information about using the ORADIM utility.
7. This step is optional if operating system authentication is used for administrative users and if SSL is used for redo transport authentication. If not, then copy the primary database's remote login password file to the appropriate directory on the far sync instance. The password file must be recopied whenever an administrative privilege (`SYSDG`, `SYSOPER`, `SYSDBA`, and so on) is granted or revoked, and after the password of any user with administrative privileges is changed.

   As of Oracle Database 12*c* Release 2 (12.2.0.1), when a password file is manually updated at a far sync instance, the redo containing the same password changes from the primary database is automatically propagated to any standby databases that are set up to receive redo from that far sync instance. The password file is updated on the standby when the redo is applied.
8. On the far sync instance site, use Oracle Net Manager to configure a listener for the far sync instance.

   See *Oracle Database Net Services Administrator's Guide* for more information about the listener.
9. On the primary system, use Oracle Net Manager to create a network service name for the far sync instance (`chicagoFS`) that is to be used by redo transport services.

   On the far sync instance system, use Oracle Net Manager to create a network service name for the primary (`chicago`) and the terminal standby (`boston`) to 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 database, the far sync instance, and the terminal standby database. The connect descriptor must also specify that a dedicated server be used.

See the *Oracle Database Net Services Administrator's Guide* for more information about service names.

10. Start the far sync instance in mount mode.
11. Verify that the far sync instance is operating properly.

    For information about validating a configuration after you create a far sync instance, see Validating a Configuration.
12. Increase the protection mode of the configuration to Maximum Availability. On the primary database, execute the following command:

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

See Also:

- Supported Protection Modes for Far Sync Instances for more information about far sync and protection modes
- Oracle Data Guard Protection Modes for more information about configuring different data protection modes

Example 5-1 Some of the Initialization Parameters Used for Far Sync Instances

Primary Database chicago

```
DB_UNIQUE_NAME=chicago

CONTROL_FILES='/arch1/chicago/control01.ctl'

DB_FILE_NAME_CONVERT='/boston/','/chicago/'

LOG_FILE_NAME_CONVERT='/boston/','/chicago/'

FAL_SERVER=boston

LOG_ARCHIVE_CONFIG='DG_CONFIG=(chicago,chicagoFS,boston)'

LOG_ARCHIVE_DEST_1='LOCATION=USE_DB_RECOVERY_FILE_DEST
VALID_FOR=(ALL_LOGFILES,ALL_ROLES) DB_UNIQUE_NAME=chicago'

LOG_ARCHIVE_DEST_2='SERVICE=chicagoFS SYNC AFFIRM
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) DB_UNIQUE_NAME=chicagoFS'
```

Far Sync Instance chicagoFS

```
DB_UNIQUE_NAME=chicagoFS

CONTROL_FILES='/arch2/chicagoFS/control01.ctl'

DB_FILE_NAME_CONVERT='/chicago/','/chicagoFS/','/boston/','/chicagoFS/'

LOG_FILE_NAME_CONVERT='/chicago/','/chicagoFS/','/boston/','/chicagoFS/'
```

```
FAL_SERVER=chicago

LOG_ARCHIVE_CONFIG='DG_CONFIG=(chicago,chicagoFS,boston)'

LOG_ARCHIVE_DEST_1='LOCATION= USE_DB_RECOVERY_FILE_DEST
VALID_FOR=(ALL_LOGFILES,ALL_ROLES) DB_UNIQUE_NAME=chicagoFS'

LOG_ARCHIVE_DEST_2='SERVICE=boston ASYNC
VALID_FOR=(STANDBY_LOGFILES,STANDBY_ROLE) DB_UNIQUE_NAME=boston'
```

Physical Standby boston

```
DB_UNIQUE_NAME=boston

CONTROL_FILES='/arch3/boston/control01.ctl'

DB_FILE_NAME_CONVERT='/chicago/','/boston/'

LOG_FILE_NAME_CONVERT='/chicago/','/boston/'

FAL_SERVER='chicagoFS','chicago'

LOG_ARCHIVE_CONFIG='DG_CONFIG=(chicago,chicagoFS,boston)'

LOG_ARCHIVE_DEST_1='LOCATION= USE_DB_RECOVERY_FILE_DEST
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'
```

5.2 Alternate Destinations

After you perform the steps in Creating and Configuring a Far Sync Instance, the far sync instance provides zero data loss capability for the configuration by forwarding the redo to the terminal standby at a remote site over the WAN. For the configuration to remain protected in the event the far sync instance is not reachable, you must configure alternate redo transport paths to the standby databases. This is accomplished using the `GROUP` and `PRIORITY` attributes of the `LOG_ARCHIVE_DEST_`*n* parameter. (As of Oracle Database 12*c* Release 2 (12.2.0.1), the `GROUP` and `PRIORITY` attributes have replaced the `ALTERNATE` attribute for remote redo destinations.)

The number of possible alternate remote destinations has been increased with the concept of log archive destination groups. A log archive destination group specifies multiple archive destinations that can be used to distribute redo to multiple destinations, either from a far sync instance or through cascading. The destinations in the group can then be prioritized so that only one destination is active at a time on the primary database. Other destinations are available to become active if the active destination becomes unavailable. To expand the number of possible archive destinations for your database, you can specify multiple groups.

> **See Also:**
>
> - Using the ALTERNATE Attribute to Configure Remote Alternate Destinations for information about configuring alternate remote destinations using the old `ALTERNATE` syntax.

5.2.1 Assigning Log Archive Destinations to a Group

Use the `GROUP` attribute of the `LOG_ARCHIVE_DEST_n` initialization parameter to assign log archive destinations to groups.

If log archive destination groups are used, then as long as at least one destination within the group remains available, at least one destination remains enabled and active. Log archive destinations that are not assigned to a group behave the same as log archive destinations did prior to Oracle Database 12*c* Release 2 (12.2.0.1).

There can be up to 30 log archive destinations in a group. Log archive destination groups are referenced by their group number, which is assigned when the group is created. Groups are numbered from 1 through 8. A log archive destination group contains a set of remote (`SERVICE=…`) destinations. (Local archival (`LOCATION=…`) destinations are not supported in log archive destination groups and must use the `ALTERNATE` attribute for alternate local archiving locations. See ALTERNATE.

One log archive destination in the group is always active and the others are available for use in the event of a failure of the active log archive destination. When a failed destination again becomes available it becomes eligible if the currently active destination fails, but it does not become active immediately, unless all other group members are also unavailable. For example, the following declaration can be used to specify three far sync instances as members of the same group and having the same priority (Priority within a group is described in the next section). These are example parameter definitions and do not contain all the necessary attributes. Do not use them verbatim. In this example only the first destination is active with the second destination available to take over if destination 1 becomes unavailable.

```
LOG_ARCHIVE_DEST_2='SERVICE=chicagoFS SYNC
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) GROUP=1'

LOG_ARCHIVE_DEST_STATE_2=ENABLE

LOG_ARCHIVE_DEST_3='SERVICE=chicagoFS1 SYNC
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) GROUP=1'

LOG_ARCHIVE_DEST_STATE_3=ALTERNATE
```

> **Note:**
>
> Because log archive destination groups replace the `LOG_ARCHIVE_DEST_n ALTERNATE` attribute, use of the `ALTERNATE` attribute with log archive destinations that are not in the default group (where `GROUP` is specified as 1 to 8) is not allowed.

5.2.2 Assigning Priorities to Log Archive Destinations in a Group

Using the PRIORITY attribute of the LOG_ARCHIVE_DEST_*n* initialization parameter to assign destination preferences within a log archive destination group allows you to control the fail back mechanism, especially with multiple members within a group.

In the previous section, the two far sync instance destinations did not have a priority, which means that when the alternate destination is activated after a failure of the first destination it remains as the active destination until it fails. The priority is used to determine which log archive destination within a group to make active when the database or far sync instance is started or when a destination fails. Log archive destinations become active in the following cases: The primary database is opened in read/write mode, a far sync instance is mounted, or a standby database is mounted or opened in read-only mode. The same priority value can be assigned to more than one log archive destination in a group. The priority value is an integer in the range of 1 through 8. Lower numbers indicate higher priorities. The default priority is 1 (the highest priority).

The priority comes into play when a previously failed destination becomes available again. A set of log archive destinations assigned to the same group have the same priority, by default. Therefore, if one destination fails then a failover occurs to another member of the set. When the failed destination becomes available again, it does not become the active destination since both destinations have the same priority. If the second destination fails after the first destination has again become available, then the database fails over to the first destination or to another destination in the group at the same priority. This cycle can repeat indefinitely, provided that another destination is always available before the active destination fails.

Continuing with the previous example, priorities can be added to the log archive destinations to control when a destination might become active. In the following example, a third far sync instance is added, but at a lower priority:

```
LOG_ARCHIVE_DEST_2='SERVICE=chicagoFS SYNC
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) GROUP=1 PRIORITY=1'

LOG_ARCHIVE_DEST_STATE_2=ENABLE

LOG_ARCHIVE_DEST_3='SERVICE=chicagoFS1 SYNC
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) GROUP=1 PRIORITY=1'

LOG_ARCHIVE_DEST_STATE_3=ALTERNATE

LOG_ARCHIVE_DEST_4='SERVICE=chicagoFS2 ASYNC
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) GROUP=1 PRIORITY=2'

LOG_ARCHIVE_DEST_STATE_4=ALTERNATE
```

This declaration results in the following behavior:

- The primary ships redo to the first of two preferred far sync instances, chicagoFS
- If chicagoFS become unavailable, then the primary ships to chicagoFS1.
- If chicagoFS becomes available again, no fail back occurs. It becomes the alternate to chicagoFS1 because the priority is the same.
- If both chicagoFS and chicagoFS1 become unavailable, then the primary ships to chicagoFS2 (in this case via the ASYNC redo transmission mode).

- If either chicagoFS or chicagoFS1 become available while the primary is shipping to chicagoFS2, then the primary fails back to that available preferred log archive destination.

5.2.3 Shipping to Multiple Active Destinations in a Group

You can also use the PRIORITY attribute to configure a group so that it ships to multiple destinations if a preferred destination fails.

The mechanism that supports multiple active destinations within a single group is that the lowest priority (PRIORITY=8) is defined to activate destinations within that group at that priority, generally used to send the redo directly to the target standby databases. The following log archive destination declaration shows how this could be configured. In this example , there is one far sync instance that forwards redo to two terminal standby databases:

```
LOG_ARCHIVE_DEST_2='SERVICE=chicagoFS SYNC
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) GROUP=1 PRIORITY=1'

LOG_ARCHIVE_DEST_STATE_2=ENABLE

LOG_ARCHIVE_DEST_3='SERVICE=boston ASYNC
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) GROUP=1 PRIORITY=8'

LOG_ARCHIVE_DEST_STATE_3=ALTERNATE

LOG_ARCHIVE_DEST_4='SERVICE=newyork ASYNC
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) GROUP=1 PRIORITY=8'

LOG_ARCHIVE_DEST_STATE_4=ALTERNATE
```

This declaration results in the following behavior:

- The primary ships redo to the preferred far sync instance, chicagoFS.
- If chicagoFS is unavailable, then the primary ships directly to both terminal standbys boston and newyork in ASYNC mode.
- While shipping to boston and newyork, if chicagoFS becomes available, then the primary stops shipping directly to boston and newyork and begins shipping instead to chicagoFS.

5.2.4 Using Multiple Log Archive Destination Groups

Multiple log archive destination groups can be used for site-specific high availability considerations or to distribute service over large cascaded (reader farm) configurations.

The following declaration sets up multiple log archive destination groups with chicagoFS and chicagoFS1 in group 1 and chicagoFS3 and chicagoFS4 in group 2:

```
LOG_ARCHIVE_DEST_2='SERVICE=chicagoFS SYNC
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) GROUP=1 PRIORITY=1'

LOG_ARCHIVE_DEST_STATE_2=ENABLE

LOG_ARCHIVE_DEST_3='SERVICE=chicagoFS1 SYNC
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) GROUP=1 PRIORITY=1'
```

```
LOG_ARCHIVE_DEST_STATE_3=ALTERNATE

LOG_ARCHIVE_DEST_4='SERVICE=chicagoFS3 SYNC
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) GROUP=2 PRIORITY=1'

LOG_ARCHIVE_DEST_STATE_4=ENABLE

LOG_ARCHIVE_DEST_5='SERVICE=chicagoFS4 SYNC
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) GROUP=2 PRIORITY=1'

LOG_ARCHIVE_DEST_STATE_5=ALTERNATE
```

5.2.5 Determining the Availability Status of Log Archive Destinations

Oracle Data Guard keeps track of the current status of available but inactive destinations in log archive destination groups by periodically polling configured destinations to determine their availability.

The information used to determine availability is derived from the `MAX_FAILURE` attribute which specifies 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. The default value for `MAX_FAILURE` is 1 when the `GROUP` and `PRIORITY` attributes are used.

The behavior of the `MAX_FAILURE` attribute is different between Oracle Database 12*c* Release 1 (12.1) and Oracle Database 12*c* Release 2 (12.2). It is important to understand the differences.

> **See Also:**
>
> - MAX_FAILURE

5.3 Configuring Alternate Destinations

Far sync instance configurations can be set up to provide varying levels of data protection.

The following topics expand on the examples provided in the previous section and provide examples of two additional far sync instance configurations that provide better data protection when you use far sync instances.

- **Reduced Protection After a Far Sync Failure**
- **Far Sync Instance High Availability**
- **Maintaining Protection After a Role Change**

5.3.1 Reduced Protection After a Far Sync Failure

With all far sync instance configurations it is important that redo continues to ship to the terminal standbys to continue to provide protection of the primary database.

In the simplest configuration there is one far sync instance (chicagoFS) and one terminal standby database (boston).

If the far sync instance fails, then redo should be shipped directly to the terminal standby by adding an additional log archive destination to the primary database, chicago. This does reduce the protection level because redo transmission is then in ASYNC mode instead of SYNC mode.

Example 5-2 Configuring for Single Destination Failover

Primary Database chicago

```
LOG_ARCHIVE_DEST_2='SERVICE=chicagoFS SYNC AFFIRM
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) DB_UNIQUE_NAME=chicagoFS GROUP=1 PRIORITY=1’

LOG_ARCHIVE_DEST_STATE_2=ENABLE

LOG_ARCHIVE_DEST_3='SERVICE=boston ASYNC NOAFFIRM
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) DB_UNIQUE_NAME=boston GROUP=1 PRIORITY=2'

LOG_ARCHIVE_DEST_STATE_3=ALTERNATE
```

This declaration causes the primary database to ship redo directly to the terminal standby if the far sync instance chicagoFS fails. If the far sync instance becomes available again, then it becomes the active destination and redo transmission goes to the far sync instance.

If the far sync instance had multiple terminal standby databases, then you would use PRIORITY=8 to ensure that all of those destinations received redo directly from the primary database if the far sync instance failed.

Example 5-3 Configuring for Multiple Standby Database Redo Destination Failover

Primary Database chicago

As in the previous example, modify the log archive destination on the primary database for the far sync instance to add it to a group with a Priority of 1 and then add a new log archive destination for each standby the far sync instance services at Priority 8 in ASYNC mode.

```
LOG_ARCHIVE_DEST_2='SERVICE=chicagoFS SYNC AFFIRM
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) DB_UNIQUE_NAME=chicagoFS GROUP=1 PRIORITY=1’

LOG_ARCHIVE_DEST_STATE_2=ENABLE

LOG_ARCHIVE_DEST_3='SERVICE=boston ASYNC NOAFFIRM
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) DB_UNIQUE_NAME=boston GROUP=1 PRIORITY=8'

LOG_ARCHIVE_DEST_STATE_3=ALTERNATE

LOG_ARCHIVE_DEST_4='SERVICE=newyork ASYNC NOAFFIRM
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) DB_UNIQUE_NAME=newyork GROUP=1 PRIORITY=8'

LOG_ARCHIVE_DEST_STATE_3=ALTERNATE
```

This declaration causes the primary database to ship redo directly to both terminal standby databases if the far sync instance chicagoFS fails. If the far sync instance becomes available again, then it becomes the active destination and redo transmission goes to the far sync instance.

5.3.2 Far Sync Instance High Availability

Configuring an alternate far sync instance keeps the protection level of the configuration at the configured protection level of Maximum Availability if the preferred far sync instance fails for some reason.

In both of the preceding examples the protection level of the configuration would fall out of Maximum Availability because redo is no longer being shipped in `SYNC` mode. For more protection from system or network failures, an additional far sync instance can be configured that provides high availability for the active far sync instance. In this configuration one is the preferred active far sync instance and the other is the alternate far sync instance.

The primary automatically starts shipping to the alternate far sync instance if it detects a failure at the preferred far sync instance. In these types of configurations, the primary uses only one far sync instance to redistribute redo at any given time.

To maintain the Maximum Availability protection level, configure two far sync instances near to the primary database and set them up to protect each other. Then, if the active far sync instance becomes unavailable, the primary database can automatically begin sending redo in synchronous mode to the alternate far sync instance, thereby maintaining the elevated protection level of Maximum Availability. In this case though, the two far sync instances have the same priority and when one takes over for the other it remains the active far sync instance until it fails. To ensure that redo continues to be shipped to the terminal standby database in the event that both far sync instances fail, the terminal standby database is configured as before with `PRIORITY=2`. (If there is more than one terminal standby database, then use `PRIORITY=8` for them).

The high availability far sync instance would be created using the same steps as given in Creating and Configuring a Far Sync Instance, and configured to forward redo to the terminal standby boston.

Example 5-4 Parameters Used to Set Up the High Availability Far Sync Instance

Primary Database chicago

```
LOG_ARCHIVE_CONFIG='DG_CONFIG=(chicago,chicagoFS,chicagoFS1,boston)'

LOG_ARCHIVE_DEST_2='SERVICE=chicagoFS SYNC AFFIRM
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) DB_UNIQUE_NAME=chicagoFS GROUP=1 PRIORITY=1’

LOG_ARCHIVE_DEST_STATE_2=ENABLE

LOG_ARCHIVE_DEST_3='SERVICE=chicagoFS1 SYNC AFFIRM
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) DB_UNIQUE_NAME=chicagoFS1 GROUP=1
PRIORITY=1’

LOG_ARCHIVE_DEST_STATE_3=ALTERNATE

LOG_ARCHIVE_DEST_4='SERVICE=boston ASYNC NOAFFIRM
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) DB_UNIQUE_NAME=boston GROUP=1 PRIORITY=2'

LOG_ARCHIVE_DEST_STATE_4=ALTERNATE
```

Oracle Data Guard can now continue synchronously sending redo to a far sync instance, maintaining the required zero data loss protection mode of Maximum Availability if a far sync instance fails. If both far sync instances fail, then redo ships in

ASYNC mode directly to boston, at a reduced protection level. As before, when either of the failed far sync instances becomes available again, Oracle Data Guard automatically resynchronizes it and returns to the original configuration, in which the primary sends redo to an active far sync instance, which then forwards that redo to the terminal standby. When the synchronization is complete, the alternate destination for the standby (LOG_ARCHIVE_DEST_4 in the preceding example) again becomes dormant as the alternate.

5.3.3 Maintaining Protection After a Role Change

These examples describe how to maintain data protection after a role change.

The configuration described in the preceding sections works well to keep the configuration running at Maximum Availability until all far sync instances fail and redo is shipped to the standby database directly. But it would be inappropriate after a role transition where boston becomes the primary database and chicago becomes the terminal standby. The far sync instances chicagoFS and chicagoFS1 would be too remote for boston to use as a synchronous destination because the network latency between two sites is sufficiently large that it would impact commit response times. To maintain the protection level of Maximum Availability for zero data loss, a second far sync instance configuration close to boston must be established, in readiness for a future role transition event.

Using the same procedure as described in Creating and Configuring a Far Sync Instance, create two far sync instances named bostonFS and bostonFS1 close to the standby database boston and configure them both to ship redo to chicago in ASYNC mode when they are active. Then add them to boston so that when boston is the primary it ships redo to one of the far sync instances in SYNC mode with all the failover capabilities that were configured for chicago and its far sync instances. You need to add the new boston far sync instances to the LOG_ARCHIVE_CONFIG on both boston and chicago.

Example 5-5 Parameters Used to Set Up Protection After a Role Change

Primary Database boston

```
LOG_ARCHIVE_CONFIG='DG_CONFIG=(chicago,chicagoFS,chicagoFS1,boston, bostonFS,
bostonFS1)'

LOG_ARCHIVE_DEST_2='SERVICE=bostonFS SYNC AFFIRM
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) DB_UNIQUE_NAME=bostonFS GROUP=1 PRIORITY=1’

LOG_ARCHIVE_DEST_STATE_2=ENABLE

LOG_ARCHIVE_DEST_3='SERVICE=bostonFS1 SYNC AFFIRM
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) DB_UNIQUE_NAME=bostonFS1 GROUP=1 PRIORITY=1’

LOG_ARCHIVE_DEST_STATE_3=ALTERNATE

LOG_ARCHIVE_DEST_4='SERVICE=chicago ASYNC NOAFFIRM
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) DB_UNIQUE_NAME=chicago GROUP=1 PRIORITY=2'

LOG_ARCHIVE_DEST_STATE_4=ALTERNATE
```

Primary Database chicago

```
LOG_ARCHIVE_CONFIG='DG_CONFIG=(chicago,chicagoFS,chicagoFS1,boston, bostonFS,
bostonFS1)'
```

Given these declarations, the far sync instance, `bostonFS`, receives redo from `boston` and ships it to `chicago` only when `boston` is the primary database. However, even if `boston` is not the primary database, Oracle recommends keeping far sync instance `bostonFS` and `bostonFS1` mounted in readiness for a future role transition.

5.4 Supported Protection Modes for Far Sync Instances

A far sync instance is supported in either maximum performance or maximum availability mode.

5.4.1 Far Sync Instances in Maximum Availability Mode Configurations

In maximum availability mode, the far sync instance is relatively close to the primary database to minimize network latency, and the primary services the far sync instance using `SYNC` transport.

> **Note:**
>
> There is no architectural limit to the distance that can separate the primary and far sync instance in maximum availability mode. The practical distance limit varies depending upon a given application's tolerance to the impact of network latency in a synchronous configuration. Also, it is possible to reduce the performance impact for any given distance by using the new Oracle Data Guard FastSync feature (`SYNC/NOAFFIRM`). See "Performance Versus Protection in Maximum Availability Mode".

Both `SYNC/AFFIRM` and `SYNC/NOAFFIRM` semantics are supported on the `LOG_ARCHIVE_DEST_`*n* established at the primary for the far sync instance. See Oracle Data Guard Protection Modes for information about the trade-offs of using each one.

When a primary services a far sync instance using `SYNC` transport, all committed redo resides on disk at the far sync instance. This allows the far sync instance to use one of the terminal standby destinations for a no data loss failover if the primary database is lost.

The far sync instance uses `ASYNC` transport to ship the incoming redo to terminal standbys that can be much farther away. This extends no data loss protection to destinations that are too far away for a primary database to feasibly service directly with `SYNC` transport because of the degradation in transaction throughput that would result. This is a case where a far sync instance is beneficial even if there is only one standby destination in the configuration.

5.4.2 Far Sync Instances in Maximum Performance Mode Configurations

In maximum performance mode, the primary database services the far sync instance destination using `ASYNC` redo transport.

This is true regardless of the physical distance between the primary and the far sync instance because high network latencies do not affect transaction throughput when a destination is serviced with `ASYNC` transport.

In maximum performance mode, a far sync instance can benefit Oracle Data Guard configurations that manage more than one remote destination. Although each `ASYNC` destination has a near-zero effect on primary database performance, if there are many remote destinations (for example, multiple Oracle Active Data Guard standbys that form a reader farm), then the effect can become measurable. When a far sync instance is used, there is zero incremental effect for each remote destination added to the configuration. Additionally, redo transport compression can also be offloaded to the far sync instance. When a far sync instance is used, the primary only has to service the far sync instance, which then services the rest of the configuration; the greater the number of destinations, the greater the performance benefit.

6
Oracle Data Guard Protection Modes

Oracle Data Guard lets you set different data protection modes.

See the following topics for information about these modes and how to set them on a primary database:

- Oracle Data Guard Protection Modes
- Setting the Data Protection Mode of a Primary Database

6.1 Oracle Data Guard Protection Modes

Oracle Data Guard provides three protection modes: maximum availability, maximum performance, and maximum protection.

In the following descriptions of the protection modes, 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 Redo Gap Detection and Resolution.

Maximum Availability

This protection mode provides the highest level of data protection that is possible without compromising the availability of a primary database. Under normal operations, transactions do not commit until all redo data needed to recover those transactions has been written to the online redo log AND based on user configuration, one of the following is true:

- redo has been received at the standby, I/O to the standby redo log has been initiated, and acknowledgement sent back to primary
- redo has been received and written to standby redo log at the standby and acknowledgement sent back to primary

If the primary does not receive acknowledgement from at least one synchronized standby, then 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.

If the primary database fails, then this mode ensures no data loss occurs provided there is at least one synchronized standby in the Oracle Data Guard configuration. See "Performance Versus Protection in Maximum Availability Mode" for information about the redo transport settings necessary to support Maximum Availability and associated trade-offs.

Transactions on the primary are considered protected as soon as Oracle Data Guard has written the redo data to persistent storage in a standby redo log file. Once that is done, acknowledgment is quickly made back to the primary database so that it can proceed to the next transaction. This minimizes the impact of synchronous transport on primary database throughput and response time. To fully benefit from complete Oracle Data Guard validation at the standby database, be sure to operate in real-time apply mode so that redo changes are applied to the standby database as fast as they

are received. Oracle Data Guard signals any corruptions that are detected so that immediate corrective action can be taken.

Performance Versus Protection in Maximum Availability Mode

When you use Maximum Availability mode, it is important to understand the possible results of using the `LOG_ARCHIVE_DEST_n` attributes `SYNC/AFFIRM` versus `SYNC/NOAFFIRM` (FastSync) so that you can make the choice best suited to your needs.

When a transport is performed using `SYNC/AFFIRM`, the primary performs write operations and waits for acknowledgment that the redo has been transmitted synchronously to the physical standby and written to disk. A `SYNC/AFFIRM` transport provides an additional protection benefit at the expense of a performance impact caused by the time required to complete the I/O to the standby redo log.

When a transport is performed using `SYNC/NOAFFIRM`, the primary performs write operations and waits only for acknowledgement that the data has been received on the standby, *not* that it has been written to disk. The `SYNC/NOAFFIRM` transport can provide a performance benefit at the expense of potential exposure to data loss in a special case of multiple simultaneous failures.

With those definitions in mind, suppose you experience a catastrophic failure at the primary site at the same time that power is lost at the standby site. Whether data is lost depends on the transport mode being used. In the case of `SYNC/AFFIRM`, in which there is a check to confirm that data is written to disk on the standby, there would be no data loss because the data would be available on the standby when the system was recovered. In the case of `SYNC/NOAFFIRM`, in which there is no check that data has been written to disk on the standby, there may be some data loss.

> **See Also:**
>
> - LOG_ARCHIVE_DEST_n Parameter Attributes for more information about the `SYNC`, `AFFIRM`, and `NOAFFIRM` attributes
> - *Oracle Data Guard Broker* for information about transporting redo in a broker configuration using `FASTSYNC` mode (using `SYNC` and `NOAFFIRM` together in maximum availability mode)

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 the time required to transmit redo data and receive acknowledgment from a standby database.

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

Maximum protection is similar to maximum availability but provides an additional level of data protection in the event of multiple failure events. Unlike maximum availability, which allows the primary to continue processing if it is unable to receive acknowledgement from a standby database, maximum protection shuts the primary database down rather than allowing it to continue processing transactions that are unprotected.

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.

> **Note:**
>
> Asynchronously committed transactions are not protected by Oracle Data Guard against loss until the redo generated by those transactions has been written to the standby redo log of at least one synchronized standby database.
>
> For more information about the asynchronous commit feature, see:
>
> - *Oracle Database Concepts*
> - *Oracle Database PL/SQL Language Reference*

6.2 Setting the Data Protection Mode of a Primary Database

Protection mode settings can be set and changed on an open database as long as the configuration meets the requirements of the protection mode (including going from maximum performance mode to maximum availability mode).

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

1. Select a data protection mode that meets your availability, performance, and data protection requirements. See Oracle Data Guard Protection Modes for a description of the data protection modes.
2. Verify that at least one standby database meets the redo transport requirements for the desired data protection mode.

   The `LOG_ARCHIVE_DEST_n` database initialization parameter that corresponds to at least one standby database must include the redo transport attributes listed in the following table for the desired data protection mode.

   The standby database must also have a standby redo log.

Table 6-1 Required Redo Transport Attributes for Data Protection Modes

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

3. Verify that the `DB_UNIQUE_NAME` database initialization parameter has been set to a unique value on the primary database and on each standby database.
4. Verify that the `LOG_ARCHIVE_CONFIG` database initialization parameter has been defined on the primary database and on each standby database, and that its value includes a `DG_CONFIG` list that includes the `DB_UNIQUE_NAME` of the primary database and each standby database.

   The following sample SQL statement configures the `LOG_ARCHIVE_CONFIG` parameter:

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

5. Set the data protection mode by executing the following SQL statement on the primary database:

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

   The data protection mode can be set to `MAXIMUM PROTECTION` on an open database only if the current data protection mode is `MAXIMUM AVAILABILITY` and if there is at least one synchronized standby database.
6. 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;
   ```

7

Redo Transport Services

An Oracle Data Guard configuration requires that Oracle redo transport services be configured and monitored.

See the following topics:

7.1 Introduction to Redo Transport Services

Redo transport services performs the automated transfer of redo data between members of an Oracle Data Guard configuration.

The following redo transport destinations are supported:

- Oracle Data Guard standby databases

  This guide describes how to create and manage physical, logical, and snapshot 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 data files, 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 data files.

- Oracle Streams downstream capture databases

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

- Far sync instances

  See Far Sync for more information about far sync instances.

- Zero Data Loss Recovery Appliance (Recovery Appliance)

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.

  Although there is no limit on the distance between a primary database and a `SYNC` redo transport destination, transaction commit latency increases as network latency increases between a primary database and a `SYNC` redo transport destination.

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

> **Note:**
>
> Synchronous redo transport is not supported for Zero Data Loss Recovery Appliance.

- 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 Oracle Data Guard Protection Modes .

7.2 Configuring Redo Transport Services

Oracle databases must be configured before they can send and receive redo data. Part of the configuration process involves setting up redo transport security.

See the following topics:

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

These topics assume that you have a thorough understanding of the following:

- Database administrator authentication
- Database initialization parameters
- Managing a redo log
- Managing archived redo logs
- Fast recovery areas
- Oracle Net Configuration

7.2.1 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.

7.2.1.1 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*, and `FAL_SERVER` 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 Security Guide* for more information about SSL
> - *Oracle Database Enterprise User Security Administrator's Guide* for more information about administering enterprise domains
> - *Oracle Label Security Administrator's Guide* for information about administering Oracle Internet Directory

7.2.1.2 Redo Transport Authentication Using a Password File

If the SSL authentication requirements are not met, then each database must use a remote login password file.

In an Oracle Data Guard configuration, all physical and snapshot standby databases must use a copy of the password file from the primary database. That copy is automatically refreshed whenever an administrative privilege (`SYSDG`, `SYSOPER`, `SYSDBA`, and so on) is granted or revoked, and after the password of any user with administrative privileges is changed. The only exception to this is far sync instances. Updated password files must still be manually copied to far sync instances because far sync instances receive redo, but do not apply it. Once the password file is up-to-date at the far sync instance the redo containing the password update at the primary is automatically propagated to any standby databases that are set up to receive redo from that far sync instance. The password file is updated on the standby when the redo is applied.

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

7.2.2 Configuring an Oracle Database to Send Redo Data

To specify a redo transport destination, use the LOG_ARCHIVE_DEST_*n* database initialization parameter (where *n* is an integer from 1 to 31).

There is a LOG_ARCHIVE_DEST_STATE_*n* database initialization parameter (where *n* is an integer from 1 to 31) that corresponds to each LOG_ARCHIVE_DEST_*n* parameter. This parameter is used to enable or disable the corresponding redo destination. Table 7-1 shows the valid values that can be assigned to this parameter.

Table 7-1 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 do not transmit redo data to this destination.
ALTERNATE	This destination becomes 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 LOG_ARCHIVE_DEST_n Parameter Attributes 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 the redo transport destination. The service name must be resolvable through an Oracle Net naming method to an Oracle Net connect descriptor that matches the Oracle Net listener(s) at the redo transport destination. The connect descriptor must specify that a dedicated server connection be used, unless that is the default connection type for the redo transport destination.

> **See Also:**
>
> *Oracle Database Net Services Administrator's Guide* for information about Oracle Net service names, connect descriptors, listeners, and network security

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

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

The `NET_TIMEOUT` attribute specifies how long the `LGWR` process waits 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 Monitoring Synchronous Redo Transport Response Time for information about monitoring synchronous redo transport mode response time.

> **Note:**
>
> You could also set the database initialization parameter, `DATA_GUARD_SYNC_LATENCY`, which is global for all synchronous standby destinations. It defines the maximum amount of time (in seconds) that the primary database may wait before disconnecting subsequent destinations after at least one synchronous standby has acknowledged receipt of the redo. For example, suppose you have three synchronous standby destinations and you set `DATA_GUARD_SYNC_LATENCY` to a value of 2. If the first standby acknowledges receipt of the redo immediately, then the primary database waits no longer than 2 seconds for the other two standbys to respond. If one or both respond within 2 seconds, then they are maintained as active destinations. Destinations that do not respond in time are marked as failed. In both cases the primary remains in zero data loss protection mode because one synchronous standby has acknowledged receipt of the redo. Any failed synchronous standbys are reconnected as normal after the number of seconds specified for the `REOPEN` attribute have passed.

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 is not 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 an Oracle Data Guard configuration so that redo transport services 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. Redo transport compression can significantly improve redo transport performance on network links with low bandwidth and high latency.

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.

The following example uses all of the LOG_ARCHIVE_DEST_*n* attributes described in this section. A DB_UNIQUE_NAME has been specified for both destinations, as has the use of compression. If a redo transport fault occurs at either destination, then redo transport attempts 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,HARTFORD)'
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=HARTFORD SYNC AFFIRM NET_TIMEOUT=30
VALID_FOR=(ONLINE_LOGFILE,PRIMARY_ROLE) REOPEN=60 COMPRESSION=ENABLE
DB_UNIQUE_NAME=HARTFORD'
LOG_ARCHIVE_DEST_STATE_3='ENABLE'
```

> **Note:**
>
> Configuration for Zero Data Loss Recovery Appliance (Recovery Appliance) is identical to configuration for any standby database. So in the preceding example, because Chicago is an ASYNC destination, it could be either a standby database or a Recovery Appliance. (Synchronous redo transport is not supported for Recovery Appliance)

7.2.2.1 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.

7.2.3 Configuring an Oracle Database to Receive Redo Data

Redo transport destination must be configured to receive and to archive redo data from a redo source database.

See the following:

- Managing Standby Redo Logs
- Cases Where Redo Is Written Directly To an Archived Redo Log File

7.2.3.1 Managing Standby Redo Logs

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 remote file server (RFS) foreground 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 ARC*n* 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 at the redo source database, for each redo thread at the redo source database. At the redo source database, query the V$LOG view to determine how many redo log groups are in the redo log at the redo source database and query the V$THREAD view to determine how many redo threads exist at 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;
```

If the redo source database is an Oracle Real Applications Cluster (Oracle RAC) or Oracle Real Application Clusters One Node (Oracle RAC One Node) database, query the V$LOG view at the redo source database to determine how many redo threads exist and specify the corresponding thread numbers when adding redo log groups to the standby redo log.

The following sample SQL statements create a standby redo log at a database that is to receive redo from a redo source database that has two redo threads:

```
SQL> ALTER DATABASE ADD STANDBY LOGFILE THREAD 1 SIZE 500M;
SQL> ALTER DATABASE ADD STANDBY LOGFILE THREAD 1 SIZE 500M;
SQL> ALTER DATABASE ADD STANDBY LOGFILE THREAD 1 SIZE 500M;
SQL> ALTER DATABASE ADD STANDBY LOGFILE THREAD 2 SIZE 500M;
SQL> ALTER DATABASE ADD STANDBY LOGFILE THREAD 2 SIZE 500M;
SQL> ALTER DATABASE ADD STANDBY LOGFILE THREAD 2 SIZE 500M;
```

Note:

Whenever a redo log group is added to a primary database, a log group must also be added to the standby redo log of each standby database in the configuration. Otherwise, the standby database may become unsynchronized after a primary log switch, which could temporarily prevent a zero data loss failover or cause a primary database operating in maximum protection mode to shut down.

7.2.3.2 Cases Where Redo Is Written Directly To an Archived Redo Log File

Redo received by a standby database is written directly to an archived redo log file if a standby redo log group is not available or if the redo was sent to resolve a redo gap. When this occurs, redo is written to the location specified by the `LOCATION` attribute of one `LOG_ARCHIVE_DEST_`*n* parameter that is valid for archiving redo received from another database. The `LOG_ARCHIVE_DEST_`*n* parameter that is used for this purpose is determined when the standby database is mounted, and this choice is reevaluated each time a `LOG_ARCHIVE_DEST_`*n* parameter is modified.

7.3 Cascaded Redo Transport Destinations

A cascaded redo transport destination (also known as a terminal destination) receives primary database redo indirectly from a standby database rather than directly from a primary database.

A physical standby database that cascades primary database redo to one or more terminal destinations at the same time it is applying changes to its local database files is known as a cascading standby database.

With cascading, the overhead associated with performing redo transport is offloaded from a primary database to a cascading standby database.

A cascading standby database can cascade primary database redo to up to 30 terminal destinations.

A cascading standby database can either cascade redo in real-time (as it is being written to the standby redo log file) or non-real-time (as complete standby redo log files are being archived on the cascading standby).

Cascading has the following restrictions:

- Only physical standby databases can cascade redo.
- Real-time cascading requires a license for the Oracle Active Data Guard option.
- Non-real-time cascading is supported on destinations 1 through 10 only. (Real-time cascading is supported on all destinations.)

> **Note:**
>
> See Before You Patch or Upgrade the Oracle Database Software for information about how to handle cascaded redo transport destinations during an Oracle Database upgrade.

Also see the following topics:

- Configuring a Terminal Destination
- Cascading Scenarios

7.3.1 Configuring a Terminal Destination

These steps describe how to configure a terminal destination.

1. Select a physical standby database to configure as a cascading standby database.
2. On the cascading standby database, configure the `FAL_SERVER` database initialization parameter with the Oracle Net alias of the primary database or of a standby database that receives redo directly from the primary database.
3. On the cascading standby database, configure a `LOG_ARCHIVE_DEST_`*n* database initialization parameter for one or more terminal destinations. Configure the `SERVICE` attribute of this destination with the Oracle Net alias of the terminal destination, and the `VALID` attribute to be valid for archival of the standby redo log while in the standby role.

   If you specify `ASYNC` transport mode on destinations 1 through 10, then redo is shipped in real-time. If you do not specify a transport mode or you specify `SYNC` on destinations 1 through 10, then redo is shipped in non-real-time. Destinations 11 through 31 operate only in `ASYNC` (real-time) transport mode.
4. At the terminal destination, configure the `FAL_SERVER` database initialization parameter with the Oracle Net alias of the cascading standby database or of another standby database that is directly connected to the primary database. Although it is also possible to specify the primary database, this would defeat the purpose of cascading, which is to reduce the redo transport overhead on the primary database.
5. Example 7-1 shows some of the database initialization parameters used by the members of an Oracle Data Guard configuration that includes a primary database named `boston` that sends redo to a local physical standby database named `boston2`, which then cascades primary database redo to a remote physical standby database named `denver`.

   A `LOG_ARCHIVE_DEST_`*n* database initialization parameter could also be configured on database `boston` that is valid for standby redo log archival to database `denver` when database `boston` is in the standby role. This would allow redo cascading to database `denver` to continue if a switchover is performed between database `boston` and database `boston2`.

Example 7-1 Some of the Initialization Parameters Used When Cascading Redo

Primary Database

```
DB_UNIQUE_NAME=boston

FAL_SERVER=boston2

LOG_ARCHIVE_CONFIG='DG_CONFIG=(boston,boston2,denver)'

LOG_ARCHIVE_DEST_1='LOCATION=USE_DB_RECOVERY_FILE_DEST
VALID_FOR=(ALL_LOGFILES,ALL_ROLES) DB_UNIQUE_NAME=boston'

LOG_ARCHIVE_DEST_2='SERVICE=boston2 SYNC
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) DB_UNIQUE_NAME=boston2'
```

Cascading Physical Standby Database

```
DB_UNIQUE_NAME=boston2

FAL_SERVER=boston

LOG_ARCHIVE_CONFIG= 'DG_CONFIG=(boston,boston2,denver)'

LOG_ARCHIVE_DEST_1='LOCATION= USE_DB_RECOVERY_FILE_DEST
VALID_FOR=(ALL_LOGFILES,ALL_ROLES) DB_UNIQUE_NAME=boston2'

LOG_ARCHIVE_DEST_2= 'SERVICE=denver
VALID_FOR=(STANDBY_LOGFILES,STANDBY_ROLE) DB_UNIQUE_NAME=denver'
```

Cascaded Physical Standby Database

```
DB_UNIQUE_NAME=denver

FAL_SERVER=boston2

LOG_ARCHIVE_CONFIG='DG_CONFIG=(boston,boston2,denver)'

LOG_ARCHIVE_DEST_1='LOCATION= USE_DB_RECOVERY_FILE_DEST
VALID_FOR=(ALL_LOGFILES,ALL_ROLES) DB_UNIQUE_NAME=denver'
```

For information about validating a configuration after you set up a cascading environment, see "Validating a Configuration".

7.3.2 Cascading Scenarios

A Data Guard configuration can be set up to cascade to a single physical standby or to multiple physical standbys.

See the following topics:

- Cascading to a Physical Standby
- Cascading to Multiple Physical Standbys

7.3.2.1 Cascading to a Physical Standby

These steps provide an example of cascading to a physical standby database.

In this scenario, you have a mission-critical primary database. This database has stringent performance and data protection requirements, so you have decided to deploy a local physical standby database to provide zero data loss protection and a

remote, cascaded physical standby database to protect against regional disasters at the primary and local standby database sites.

You can achieve the objectives described above by performing the following steps:

1. Create a physical standby database at a local site.
2. Create a physical standby database at a site that is sufficiently remote to provide protection against regional disasters at the primary and local standby database sites.
3. Configure the local standby database as a `SYNC` redo transport destination of the primary database.
4. Configure the remote physical standby database as a terminal destination of the local standby database.

7.3.2.2 Cascading to Multiple Physical Standbys

These steps provide an example of cascading to multiple physical standby databases.

In this scenario, you have a primary database in North America and you want to deploy three replicas of this database in Europe to support read-only reporting applications. For cost and performance reasons, you do not want to maintain network links from North America to each of your European sites.

You can achieve the objectives described above by performing the following steps:

1. Create a network link between your North American site and one of your European sites.
2. Create a physical standby database at each of your European sites.
3. Open your physical standby databases in real-time query mode, as described in Opening a Physical Standby Database .
4. Configure the physical standby database at the European endpoint of your transatlantic network link to cascade redo to your other European standby databases.
5. Configure the other two physical standby databases as terminal destinations of the cascading standby database configured in step 4.

7.4 Data Protection Considerations for Cascading Standbys

When your configuration includes cascading standbys, each destination should have a `LOG_ARCHIVE_DEST_`*n* parameter defined that points back to its source for use during a failover.

Real-time cascade enables a cascaded standby database to provide nearly the same level of data protection as any standby database that receives redo directly from a primary database using asynchronous redo transport. However, although redo is forwarded in real-time, the fact that there is a second network hop creates the potential for additional data loss if an outage prevents all redo from reaching the terminal destination.

7.5 Validating a Configuration

To validate an Oracle Data Guard configuration after you create it, query the `V$DATAGUARD_CONFIG` view from any database in the configuration.

The view displays the unique database names defined with the `DB_UNIQUE_NAME` and `LOG_ARCHIVE_CONFIG` initialization parameters.

> **See Also:**
>
> - *Oracle Database Reference* for more information about the `V$DATAGUARD_CONFIG` view

7.6 Monitoring Redo Transport Services

You can monitor redo transport status, as well as redo transport response time.

See the following topics:

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

7.6.1 Monitoring Redo Transport Status

You can query views to monitor redo transport status on a redo source database.

Take the following steps to monitor redo transport status on a redo source database.

1. 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$ARCHIVED_LOG -
> WHERE RESETLOGS_CHANGE# = (SELECT MAX(RESETLOGS_CHANGE#) FROM V$ARCHIVED_LOG) -
> GROUP BY THREAD#;
```

2. 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# -
> FROM V$ARCHIVE_DEST_STATUS -
> 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.

3. Perform a query 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 -
> (SELECT THREAD#, SEQUENCE# FROM V$ARCHIVED_LOG WHERE DEST_ID=1) -
> LOCAL WHERE -
> LOCAL.SEQUENCE# NOT IN -
> (SELECT SEQUENCE# FROM V$ARCHIVED_LOG WHERE DEST_ID=2 AND -
> THREAD# = LOCAL.THREAD#);

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

4. 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 Setting Archive Tracing for complete details and examples.

7.6.2 Monitoring Synchronous Redo Transport Response Time

The V$REDO_DEST_RESP_HISTOGRAM view contains response time data for each redo transport destination.

The 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 -
> 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 -
> 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 -
> 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.

7.6.3 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 is used to specify that redo data be compressed before transmission to the destination.

- 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 LOG_ARCHIVE_DEST_n Parameter Attributes for more information about the `COMPRESSION` and `MAX_CONNECTIONS` attributes.

7.6.3.1 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 -
> 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, then this query does not return any rows, although there is 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, then the query shows 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 -
> WHERE NEXT_CHANGE# NOT IN -
> (SELECT FIRST_CHANGE# FROM DBA_LOGSTDBY_LOG WHERE L.THREAD# = THREAD#) -
> 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.

7.6.4 Redo Transport Services Wait Events

You can use Oracle wait events to track redo transport wait time on a redo source database.

Table 7-2 lists several of these Oracle wait events, which 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://www.oracle.com/goto/maa

Table 7-2 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

7.7 Tuning Redo Transport

You can optimize redo transport for best performance.

See the Oracle Data Guard Redo Transport and Network Configuration Best Practices white paper available on the Oracle Maximum Availability Architecture (MAA) home page at:

http://www.oracle.com/goto/maa

8
Apply Services

These concepts describe how redo data is applied to a standby database.

- Introduction to Apply Services
- Apply Services Configuration Options
- Applying Redo Data to Physical Standby Databases
- Applying Redo Data to Logical Standby Databases
- Standby Considerations When Removing or Renaming a PDB at a Primary

8.1 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 waits for a standby redo log file to be archived before applying the redo that it contains. However, you can enable real-time apply, which allows apply services to apply the redo in the current standby redo log file as it is being filled. Real-time apply is described in more detail in Using Real-Time Apply to Apply Redo Data Immediately.

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.

8.2 Apply Services Configuration Options

You can apply redo data immediately or you can specify a time delay to apply archived redo log files.

See the following topics:

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

8.2.1 Using Real-Time Apply to Apply Redo Data Immediately

If the real-time apply feature is enabled, then 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 already been applied to the standby database by the time the failover or switchover begins. It also enables real-time reporting on an Oracle Active Data Guard standby by keeping it more closely synchronized with the primary database.

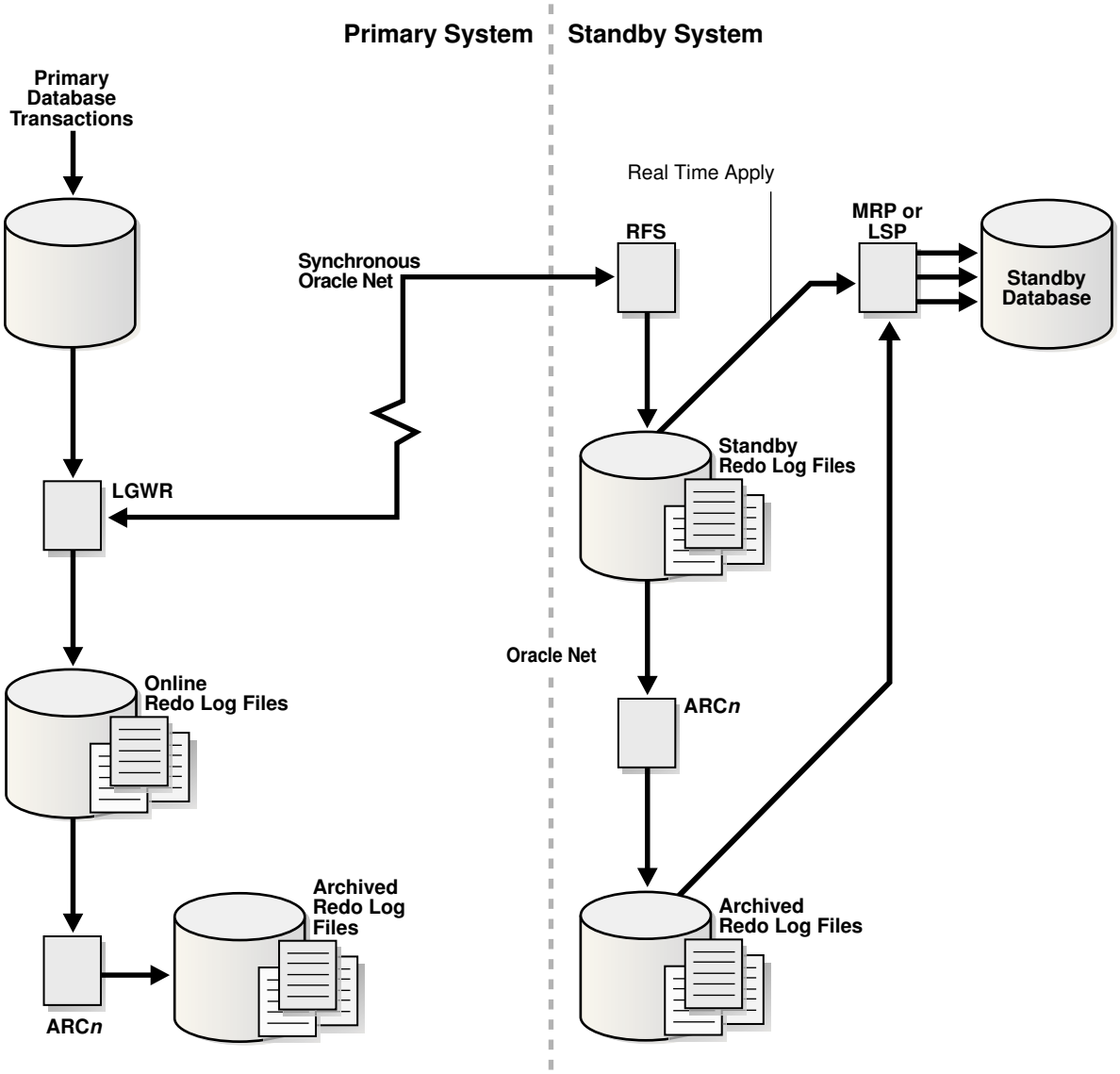
Use the `ALTER DATABASE` statement to enable the real-time apply feature, as follows:

- For physical standby databases, issue the `ALTER DATABASE RECOVER MANAGED STANDBY DATABASE` statement. (As of Oracle Database 12*c* Release 1 (12.1), the `USING CURRENT LOGFILE` clause is deprecated and no longer necessary to start real-time apply.)
- For logical standby databases, issue the `ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE` statement.

Real-time apply requires a standby database that is configured with a standby redo log and that is in ARCHIVELOG mode.

Figure 8-1 shows an Oracle 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 8-1 Applying Redo Data to a Standby Destination Using Real-Time Apply



8.2.2 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. If you define a delay as described in the following paragraph, then you must start the apply using the `USING ARCHIVED LOGFILE` clause as shown in Starting Redo Apply .

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.

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

8.3 Applying Redo Data to Physical Standby Databases

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 remote file server (RFS) process.

This section contains the following topics:

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

8.3.1 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.

Start apply services on a physical standby database as follows:

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

This also automatically enables real-time apply provided the standby database is configured with a standby redo log and is in `ARCHIVELOG` mode.

Redo Apply can be run either as a foreground session or as a background process. 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;
```

or

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE USING ARCHIVED LOGFILE
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 command does not disconnect the current SQL session.

8.3.2 Stopping Redo Apply

Use an `ALTER DATABASE` SQL statement to stop Redo Apply.

For example, issue the following SQL statement:

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

8.3.3 Monitoring Redo Apply on Physical Standby Databases

You can monitor the status of apply services on a physical standby database.

See Using Views to Monitor Primary_ Physical_ and Snapshot Standby Databases. You can also monitor the standby database using Oracle Enterprise Manager Cloud Control.

8.4 Applying Redo Data to Logical Standby Databases

SQL Apply converts the data from the archived redo log or standby redo log into SQL statements and then executes these SQL statements on the logical standby database.

Because the logical standby database remains open, tables that are maintained can be used simultaneously for other tasks such as reporting, summations, and queries.

See the following topics:

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

8.4.1 Starting SQL Apply

Use an `ALTER DATABASE` SQL statement to start SQL Apply.

For example, 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;
```

8.4.2 Stopping SQL Apply on a Logical Standby Database

Use an `ALTER DATABASE` SQL statement to stop SQL Apply on a logical standby database.

For example, 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.

8.4.3 Monitoring SQL Apply on Logical Standby Databases

There are views that provide information you can use to manage and monitor SQL Apply on logical standby databases.

See Views Related to Managing and Monitoring a Logical Standby Database. You can also monitor the standby database using Oracle Enterprise Manager Cloud Control. See Troubleshooting Oracle Data Guard.

8.5 Standby Considerations When Removing or Renaming a PDB at a Primary

Restrictions apply when you are removing or renaming a pluggable database (PDB) at the primary, if the primary is a multitenant container database (CDB).

- To perform DDL `UNPLUG` and `DROP` operations on a PDB, the PDB must first be closed on the primary as well as on all standby databases.

- To perform a DDL `RENAME` operation on a PDB, the PDB must first be put in open restricted mode on the primary, and closed on all standby databases.

If you do not close the PDB at the standby before removing it or renaming it at the primary database, then the standby stops the recovery process for all PDBs. You must close the dropped PDB at the standby and then restart recovery using the following SQL statement:

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

9
Role Transitions

An Oracle Data Guard configuration consists of one database that functions in the primary role and one or more databases that function in the standby role.

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 an Oracle 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.

See the following topics for information about how to manage role transitions in an Oracle Data Guard configuration:

- Introduction to Role Transitions
- Role Transitions Involving Physical Standby Databases
- Role Transitions Involving Logical Standby Databases
- Using Flashback Database After a Role Transition

> **Note:**
>
> These topics describe how to perform role transitions manually, using SQL statements. Do not use these manual procedures to perform role transitions in an Oracle Data Guard configuration that is managed by the broker. Use the role transition procedures provided in *Oracle Data Guard Broker* instead.

> **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 Cloud Control 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 Oracle 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.

9.1 Introduction to Role Transitions

A database operates in one of the following mutually exclusive roles: **primary** or **standby**.

Oracle Data Guard enables you to change these roles dynamically by using SQL statements, or by using either of the Oracle 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 Oracle 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.

See Also:

- Preparing for a Role Transition for information that helps you choose the role transition that best minimizes downtime and risk of data loss
- Switchovers for more information about switchovers.
- Failoversfor more information about failovers
- *Oracle Data Guard Broker* for information about event notification and database connection failover support available to database clients when a broker-managed failover occurs

9.1.1 Preparing for a Role Transition

Before starting any role transitions, you must verify that each database is properly configured and that there are no redo transport errors or redo gaps at the standby database.

- Verify that each database is properly configured for the role that it is about to assume. See Creating a Physical Standby Database and 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.

- Verify that there are no redo transport errors or redo gaps at the standby database by querying the V$ARCHIVE_DEST_STATUS view on the primary database.

  For example, the following query would be used to check the status of the standby database associated with LOG_ARCHIVE_DEST_2:

  ```
  SQL> SELECT STATUS, GAP_STATUS FROM V$ARCHIVE_DEST_STATUS WHERE DEST_ID = 2;

  STATUS GAP_STATUS
  --------- ------------------------
  VALID NO GAP
  ```

  Do not proceed until the value of the STATUS column is VALID and the value of the GAP_STATUS column is NOGAP, for the row that corresponds to the standby 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 is set to become the new primary database. Not removing the delay results in a longer switchover time, and may cause the switchover to be disallowed.
- Before performing a switchover to a physical standby database that is in real-time query mode, consider bringing all instances of that standby database to the mounted but not open state to achieve the fastest possible role transition and to cleanly terminate any user sessions connected to the physical standby database prior to the role transition.
- When you perform a switchover from an Oracle RAC primary database to a physical standby database, *it is not necessary* to shut down all but one primary database instance.

9.1.2 Choosing a Target Standby Database for a Role Transition

For an Oracle 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 takes to perform the role transition. This is affected by how far behind the standby database is in applying 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 behave after the role transition. If the new primary was a physical standby before the role transition, then all other standby databases in the configuration 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 become standbys of the new primary, but physical standbys in the configuration continue to be standbys of the old primary and therefore, do not protect the new primary. In the latter case, a future switchover or failover back to the original primary database returns 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. To use a snapshot standby database as a target for a role transition, first convert it to a physical standby database and allow all redo received from the primary database to be applied. See Converting a Snapshot Standby Database into a Physical Standby Database.

Oracle Data Guard provides the `V$DATAGUARD_STATS` view, which you can use to evaluate each standby database in terms of the currency of the data in the standby database, and the time needed to perform a role transition if all available redo data is applied to the standby database. For example:

```
SQL> COLUMN NAME FORMAT A24
SQL> COLUMN VALUE FORMAT A16
SQL> COLUMN DATUM_TIME FORMAT A24
SQL> SELECT NAME, VALUE, DATUM_TIME FROM V$DATAGUARD_STATS;

NAME                     VALUE            DATUM_TIME
------------------------ ---------------- ------------------------
transport lag            +00 00:00:00     06/18/2009 12:22:06
apply lag                +00 00:00:00     06/18/2009 12:22:06
apply finish time        +00 00:00:00.000
estimated startup time   9
```

This query output shows that the standby database has received and applied all redo generated by the primary database. These statistics were computed using data received from the primary database as of 12:22.06 on 06/18/09.

The `apply lag` and `transport lag` metrics are computed based on data received from the primary database. These metrics become stale if communications between the primary and standby database are disrupted. An unchanging value in the `DATUM_TIME` column for the `apply lag` and `transport lag` metrics indicates that these metrics are not being updated and have become stale, possibly due to a communications fault between the primary and standby databases.

9.1.3 Switchovers

A switchover is typically used to reduce primary database downtime during planned outages.

Planned outages are events such as operating system or hardware upgrades, or rolling upgrades of the Oracle database software and patch sets.

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 9-1 shows a two-site Oracle 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 9-1 Oracle Data Guard Configuration Before Switchover

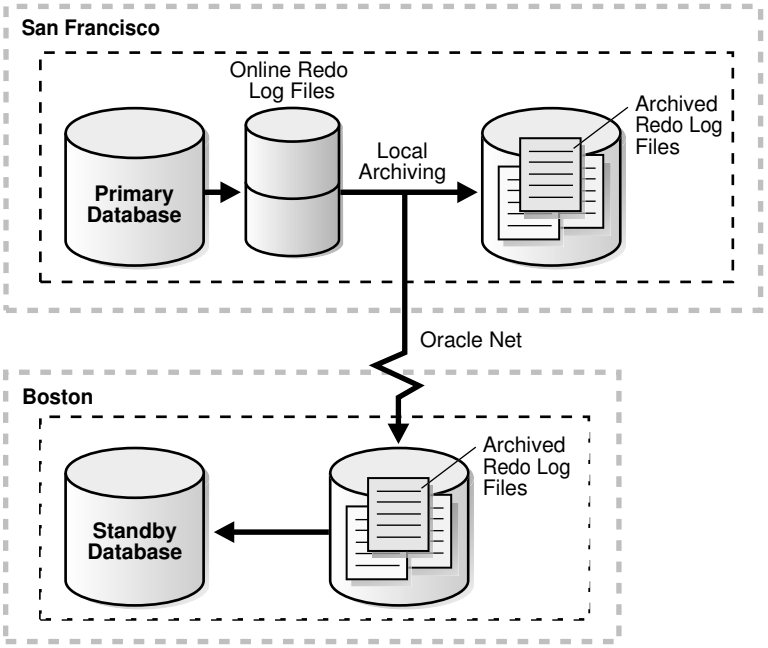


Figure 9-2 shows the Oracle Data Guard environment after the original primary database was switched over to a standby database, but before the original standby database has become the new primary database. At this stage, the Oracle Data Guard configuration temporarily has two standby databases.

Figure 9-2 Standby Databases Before Switchover to the New Primary Database

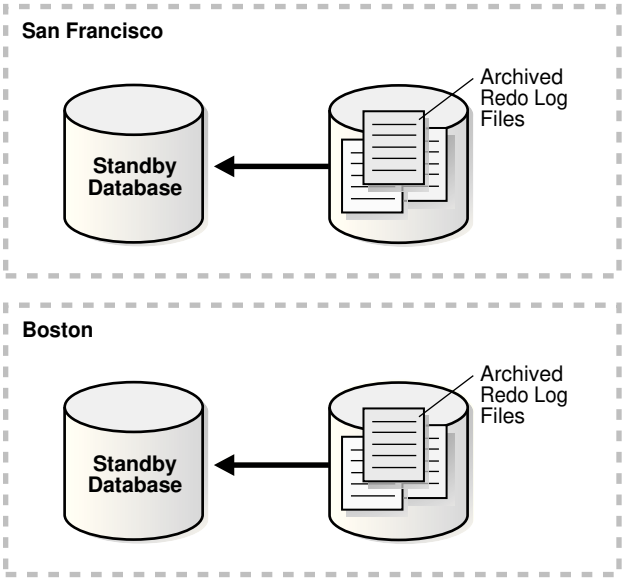
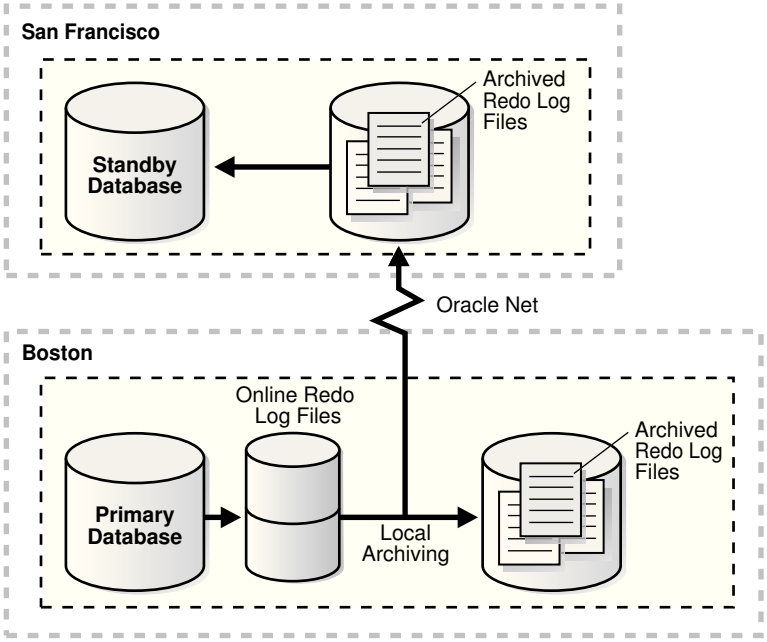


Figure 9-3 shows the Oracle Data Guard environment after a switchover took place. The original standby database became the new primary database. The primary database is now in Boston, and the standby database is now in San Francisco.

Figure 9-3 Oracle Data Guard Environment After Switchover



Preparing for a Switchover

Ensure the prerequisites listed in Preparing for a Role Transition 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.
- For switchovers involving a logical standby database, verify that both the primary and standby database instances are open and that SQL Apply is active.

> **See Also:**
>
> - Using SQL Apply to Upgrade the Oracle Database
> - Applying Redo Data to Physical Standby Databases for more information about Redo Apply
> - Applying Redo Data to Logical Standby Databases for more information about SQL Apply

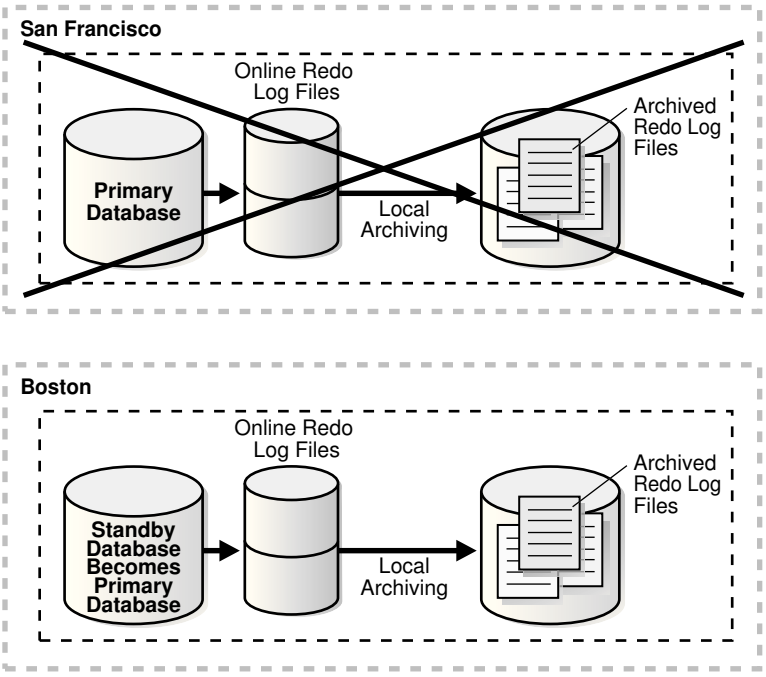
9.1.4 Failovers

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 Oracle Data Guard configuration at the time of the failover, and on the specific SQL statements used to initiate the failover.

Figure 9-4 shows the result of a failover from a primary database in San Francisco to a physical standby database in Boston.

Figure 9-4 Failover to a Standby Database



Preparing for a Failover

> **Note:**
>
> If managed standby recovery at a physical standby database chosen for failover has stopped with error `ORA-752` or `ORA-600 [3020]`, then proceed directly to Recovering From Lost-Write Errors on a Primary Database.

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

Ensure the prerequisites listed in Preparing for a Role Transition are satisfied. In addition, the following prerequisites must be met for a failover:

- If a standby database currently running in maximum protection mode is involved in the failover, then 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, then issuing the `ALTER DATABASE` statement to change the standby database from maximum protection mode to maximum performance mode does not succeed. Because a failover removes the original primary database from the Oracle 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 Verify the Physical Standby Database Is Performing Properly
>
> – See Verify the Logical Standby Database Is Performing Properly

9.1.5 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.

9.2 Role Transitions Involving Physical Standby Databases

The procedures to perform switchovers and failovers to a physical standby database have been simplified if you are running Oracle Database 12*c* Release 1 (12.1) or later.

The former procedures are still supported, however Oracle recommends that you use the new procedures as described in the following sections:

- Performing a Switchover to a Physical Standby Database
- Performing a Failover to a Physical Standby Database

Keeping Physical Standby Sessions Connected During Role Transition

As of Oracle Database 12c Release 2 (12.2.0.1), when a physical standby database is converted into a primary you have the option to keep any sessions connected to the physical standby connected, without disruption, during the switchover/failover.

To enable this feature, set the `STANDBY_DB_PRESERVE_STATES` initialization parameter in your init.ora file before the standby instance is started. This parameter applies to physical standby databases only. The allowed values are:

- `NONE` — No sessions on the standby are retained during a switchover/failover. This is the default value.
- `ALL` — User sessions are retained during switchover/failover.
- `SESSION` — User sessions are retained during switchover/failover.

> **See Also:**
>
> - Troubleshooting Oracle Data Guard for information about how to troubleshoot problems you might encounter when performing role transitions to a physical standby database
> - Performing Role Transitions Using Old Syntax for information about the procedures used in prior releases, and a comparison of old and new syntax
> - *Oracle Database Reference* for a complete description of the `STANDBY_DB_PRESERVE_STATES` initialization parameter.

9.2.1 Performing a Switchover to a Physical Standby Database

These steps describe how to perform a switchover to a physical standby database.

> **Note:**
>
> If there is a far sync instance (or a combination of preferred and alternate far sync instances) connecting the primary and standby databases, then the procedure to switchover to the standby is the same as described in this topic. Whether the far sync instance(s) are available or unavailable does not affect switchover. During switchover, the primary and standby must be able to communicate directly with each other and perform the switchover role transition steps oblivious of the far sync instance(s). See Far Sync for examples of how to set up such configurations correctly so that the far sync instance(s) can service the new roles of the two databases after switchover.

1. Verify that the target standby database is ready for switchover.

   The new switchover statement has a `VERIFY` option that results in checks being performed of many conditions required for switchover. Some of the items checked are: whether Redo Apply is running on the switchover target; whether the release version of the switchover target is 12.1 or later; whether the switchover target is synchronized; and whether it has MRP running.

   Suppose the primary database has a `DB_UNIQUE_NAME` of `BOSTON` and the switchover target standby database has a `DB_UNIQUE_NAME` of `CHICAGO`. On the primary database `BOSTON`, issue the following SQL statement to verify that the switchover target, `CHICAGO`, is ready for switchover:

   ```
   SQL> ALTER DATABASE SWITCHOVER TO CHICAGO VERIFY;
   ERROR at line 1:
   ORA-16470: Redo Apply is not running on switchover target
   ```

   If this operation had been successful, a `Database Altered` message would have been returned but in this example an `ORA-16470` error was returned. This error means that the switchover target `CHICAGO` is not ready for switchover. Redo Apply must be started before the switchover operation.

   After Redo Apply is started, issue the following statement again:

   ```
   SQL> ALTER DATABASE SWITCHOVER TO CHICAGO VERIFY;
   ERROR at line 1:
   ORA-16475: succeeded with warnings, check alert log for more details
   ```

   The switchover target, `CHICAGO`, is ready for switchover. However, the warnings indicated by the `ORA-16475` error may affect switchover performance. The alert log contains messages similar to the following:

   ```
   SWITCHOVER VERIFY WARNING: switchover target has dirty online redo logfiles that
   require clearing. It takes time to clear online redo logfiles. This may slow
   down switchover process.
   ```

   You can fix the problems or if switchover performance is not important, those warnings can be ignored. After making any fixes you determine are necessary, issue the following SQL statement again:

```
SQL> ALTER DATABASE SWITCHOVER TO CHICAGO VERIFY;
Database altered.
```

The switchover target, CHICAGO, is now ready for switchover.

2. Initiate the switchover on the primary database, BOSTON, by issuing the following SQL statement:

```
SQL> ALTER DATABASE SWITCHOVER TO CHICAGO;
Database altered.
```

If this statement completes without any errors, proceed to Step 3.

If an error occurs, mount the old primary database (BOSTON) and the old standby database (CHICAGO). On both databases, query DATABASE_ROLE from V$DATABASE. There are three possible combinations of database roles for BOSTON and CHICAGO. The following table describes these combinations and provides the likely cause and a high level remedial action for each situation. For details on specific error situations, see Troubleshooting Oracle Data Guard.

Value of DATABASE_ROLE column in V$DATABASE	Cause and Remedial Action
BOSTON database is primary, CHICAGO database is standby	Cause: The BOSTON database failed to convert to a standby database role. Action: See the alert log for details on the error that prevented BOSTON from switching to a standby role, take the necessary actions to fix the error, reopen one of the nodes of BOSTON if necessary, and repeat the switchover process from Step 1.

BOSTON database is standby, CHICAGO database is standby	Cause: The CHICAGO database failed to convert to a primary database role.
	Action: Issue the following SQL statement to convert either BOSTON or CHICAGO to a primary database:
	`SQL> ALTER DATABASE SWITCHOVER TO target_db_name FORCE;`
	For example:
	• On the CHICAGO database, issue the following SQL statement to convert it to a primary database:
	`ALTER DATABASE SWITCHOVER TO CHICAGO FORCE;`
	• On the BOSTON database, issue the following SQL statement to convert it to a primary database:
	`ALTER DATABASE SWITCHOVER TO BOSTON FORCE;`
	If the SQL statement fails with an ORA-16473 error, then you must start Redo Apply before reissuing the command.
	Restart Redo Apply as follows:
	`SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE DISCONNECT;`
	Reissue the switchover command as follows:
	`SQL> ALTER DATABASE SWTICHOVER TO BOSTON FORCE;`
	`Database altered.`

Value of DATABASE_ROLE column in V$DATABASE	Cause and Remedial Action
BOSTON database is standby, CHICAGO database is primary	Cause: The BOSTON and CHICAGO databases have successfully switched to their new roles, but there was an error communicating the final success status back to BOSTON. Action: Continue to Step 3 to finish the switchover operation.

3. Issue the following SQL statement on the new primary database, CHICAGO, to open it.

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

4. Issue the following SQL statement to mount the new physical standby database, BOSTON:

```
SQL> STARTUP MOUNT;
```

   Or, if BOSTON is an Oracle Active Data Guard physical standby database, then issue the following SQL statement to open it read only:

```
SQL> STARTUP;
```

5. Start Redo Apply on the new physical standby database. For example:

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

9.2.2 Performing a Failover to a Physical Standby Database

These steps describe how to perform a failover to a physical standby database.

1. If the primary database can be mounted, then flush any unsent archived and current redo from the primary database to the standby database. If this operation is successful, a zero data loss failover is possible even if the primary database is not in a zero data loss data protection mode.

   First, ensure that Redo Apply is active at the target standby database. Then mount, but do not open the primary database. If the primary database cannot be mounted, go to Step 2.

   If not already done, then set up the remote LOG_ARCHIVE_DEST_*n* configured at the primary to point to the target destination. (You may not have any remote LOG_ARCHIVE_DEST_*n* configured if the target destination was serviced by a far sync instance, or was a terminal standby in a cascaded configuration.) Also, ensure that the primary can connect to the target destination by verifying that the NET_ALIAS_TARGET_DB_NAME is valid and properly established.

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_6='SERVICE=NET_ALIAS_TARGET_DB_NAME -
> ASYNC VALID_FOR=(online_logfile, primary_role) -
> DB_UNIQUE_NAME="target_db_unique_name"' SCOPE=memory;

SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_6=ENABLE;
```

   It is also assumed that the LOG_ARCHIVE_CONFIG specification includes the DB_UNIQUE_NAME of the target destination at the primary (and LOG_ARCHIVE_CONFIG at the target destination includes the DB_UNIQUE_NAME of the primary). If not, then add that information to the LOG_ARCHIVE_CONFIG at the primary and target destination as required.

Issue the following SQL statement at the primary database:

```
SQL> ALTER SYSTEM FLUSH REDO TO target_db_name;
```

For `target_db_name`, specify the `DB_UNIQUE_NAME` of the standby database that is to receive the redo flushed from the primary database.

This statement flushes any unsent redo from the primary database to the standby database, and waits for that redo to be applied to the standby database.

If this statement completes without any errors, go to Step 5.If the statement completes with any error, or if it must be stopped because you cannot wait any longer for the statement to complete, continue with Step 2.

2. Query the `V$ARCHIVED_LOG` view on the target standby database to obtain the highest log sequence number for each redo thread.

   For example:

```
SQL> SELECT UNIQUE THREAD# AS THREAD, MAX(SEQUENCE#) -
> OVER (PARTITION BY thread#) AS LAST from V$ARCHIVED_LOG;

    THREAD       LAST
---------- ----------
         1        100
```

   If possible, copy the most recently archived redo log file for each primary database redo thread to the standby database if it does not exist there, and register it. This must be done for each redo thread.

   For example:

```
SQL> ALTER DATABASE REGISTER PHYSICAL LOGFILE 'filespec1';
```

3. Query the `V$ARCHIVE_GAP` view on the target standby database 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 sequence numbers 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 at the target standby database. This must be done for each redo thread.

   For example:

```
SQL> ALTER DATABASE REGISTER PHYSICAL LOGFILE 'filespec1';
```

4. The query executed in Step 3 displays information for the highest gap only. After resolving a gap, you must repeat the query until no more rows are returned.

   If, after performing Step 2 through Step 4, 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), then you can expect some data loss during the failover.

5. Issue the following SQL statement on the target standby database:

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

6. Issue the following SQL statement on the target standby database:

   ```
   SQL> ALTER DATABASE FAILOVER TO target_db_name;
   ```

   For example, suppose the target standby database is named `CHICAGO`:

   ```
   SQL> ALTER DATABASE FAILOVER TO CHICAGO;
   ```

   If this statement completes without any errors, proceed to Step 10.

   If there are errors, go to Step 7.

7. If an error occurs, try to resolve the cause of the error and then reissue the statement.
   - If successful, go to Step 10.
   - If the error still occurs and it involves a far sync instance, go to Step 8.
   - If the error still occurs and there is no far sync instance involved, go to Step 9.

8. This step is for far sync instance error cases only. If the error involves a far sync instance (for example, it is unavailable) and you have tried resolving the issue and reissuing the statement without success, then you can use the `FORCE` option. For example:

   ```
   SQL> ALTER DATABASE FAILVOVER TO CHICAGO FORCE;
   ```

   The `FORCE` option instructs the failover to ignore any failures encountered when interacting with the far sync instance and proceed with the failover, if at all possible. (The `FORCE` option has meaning only when the failover target is serviced by a far sync instance.)

   If the `FORCE` option is successful, go to Step 10.

   If the `FORCE` option is unsuccessful, go to Step 9.

9. Perform a data loss failover.

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

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

   In the following example, the failover operation fails with an `ORA-16472` error. That error means the database is configured in MaxAvailability or MaxProtection mode but data loss is detected during failover.

   ```
   SQL> ALTER DATABASE FAILOVER TO CHICAGO;
   ERROR at line 1:
   ORA-16472: failover failed due to data loss
   ```

   You can complete the data loss failover by issuing the following SQL statement:

   ```
   SQL> ALTER DATABASE ACTIVATE PHYSICAL STANDBY DATABASE;
   Database altered.
   ```

10. Open the new primary database:

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

11. Oracle recommends that you perform a full backup of the new primary database.

12. If Redo Apply has stopped at any of the other physical standby databases in your Data Guard configuration, then restart it. For example:

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

13. 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 Converting a Failed Primary Into a Standby Database Using Flashback Database or Converting a Failed Primary into a Standby Database Using RMAN Backups, or it can be re-created as a physical standby database from a backup of the new primary database using the method described in Step-by-Step Instructions for Creating a Physical Standby Database.

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

9.3 Role Transitions Involving Logical Standby Databases

Role transition steps differ depending on whether you are performing a switchover or a failover.

See the following topics for information on how to perform switchovers and failovers involving a logical standby database:

- Performing a Switchover to a Logical Standby Database
- Performing a Failover to a Logical Standby Database

> **Note:**
>
> Logical standby does not replicate database services. In the event of a failover or switchover to a logical standby, mid-tiers connecting to services in the primary are not able to connect (since the creation of the service is not replicated), or connect to an incorrect edition (since the modification of the service attribute is not replicated).
>
> Oracle Clusterware does not replicate the services it manages to logical standbys. You must manually keep them synchronized between the primary and standby. See *Oracle Clusterware Administration and Deployment Guide* for more information about Oracle Clusterware.

9.3.1 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.

For the switchover to succeed, these steps must be performed in the order in which they are described.

1. 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 Oracle 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.

2. 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.

3. 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 Oracle Data Guard configuration. The sites receiving redo data from this logical standby database accept the redo data but they do not apply it.

   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`.

4. 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 must 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:
>
> a. Cancel switchover on the primary database:
>
> ```
> SQL> ALTER DATABASE PREPARE TO SWITCHOVER CANCEL;
> ```
>
> b. Cancel the switchover on the logical standby database:
>
> ```
> SQL> ALTER DATABASE PREPARE TO SWITCHOVER CANCEL;
> ```

5. 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, prevents any new users from starting new transactions, and establishes a point in time for the switchover to be committed.

   Executing this statement also prevents 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.

6. 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, 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.

7. 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 Oracle Data Guard configuration. As described in Choosing a Target Standby Database for a Role Transition, all other logical standbys in the configuration become standbys of the new primary, but any physical standby databases remain standbys of the original primary database.

8. On the new logical standby database, start SQL Apply:

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

9.3.2 Performing a Failover to 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.

This topic describes how to perform failovers involving a logical standby database. 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 Converting a Failed Primary Into a Standby Database Using Flashback Database 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.

1. If the primary database can be mounted, then flush any unsent archived and current redo from the primary database to the standby database. If this operation is successful, a zero data loss failover is possible even if the primary database is not in a zero data loss data protection mode.

   First, ensure that Redo Apply is active at the target standby database. Then mount, but do not open the primary database.

   Issue the following SQL statement at the primary database:

   ```
   SQL> ALTER SYSTEM FLUSH REDO TO target_db_name;
   ```

   For `target_db_name`, specify the `DB_UNIQUE_NAME` of the standby database that is to receive the redo flushed from the primary database.

   This statement flushes any unsent redo from the primary database to the standby database, and waits for that redo to be applied to the standby database.

2. 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:

   a. Determine if any archived redo log files are missing on the logical standby database.

   b. Copy missing log files from the primary database to the logical standby database.

c. Register the copied log files.

You can register an archived redo log file with the logical standby database by issuing the following statement, for example:

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

3. 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 that this change persists 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.

4. 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 remote file server (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 is not applied before the standby database becomes the primary database.

5. Follow the method described in Configuring Logical Standby Databases After a Failover to ensure existing logical standby databases can continue to provide protection for the new primary database.

6. Back up the new primary database immediately after the Oracle 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.

7. 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 Converting a Failed Primary Into a Standby Database Using Flashback Database, or it can be recreated as a logical standby database from a backup of the new primary database as described in Creating a Logical Standby Database .

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

9.4 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.

If you flash back a primary database, you must flash back all of its standby databases to either the same (or earlier) SCN or time.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

9.4.1 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. 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 still runs 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.

9.4.2 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 Converting a Failed Primary Into a Standby Database Using Flashback Database for the complete step-by-step procedure.

10

Managing Physical and Snapshot Standby Databases

This chapter discusses the various ways that physical and snapshot standby databases need to be managed.

See the following topics:

- 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
- Tuning Databases in an Active Data Guard Environment with SQL Tuning Advisor
- Using Oracle Diagnostic Pack to Tune Oracle Active Data Guard Standbys
- Managing a Snapshot Standby Database

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

10.1 Starting Up and Shutting Down a Physical Standby Database

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

10.1.1 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.

After it has been mounted or opened, a physical standby database can receive redo data from the primary database.

See Applying Redo Data to Physical Standby Databases for information about Redo Apply and Opening a Physical Standby Database 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.

10.1.2 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.

10.2 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.

> **Note:**
>
> A physical standby database that is opened in read-only mode is subject to the same restrictions as any other Oracle database opened in read-only mode. For more information, see *Oracle Database Administrator's Guide*.

If a license for the Oracle Active Data Guard option has been purchased, Redo Apply can be active while the physical standby database is open, thus allowing queries to return results that are identical to what would be returned from the primary database. This capability is known as the real-time query feature. See Real-time query 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 stopped or restarted in a mounted state before starting Redo Apply.

> **See Also:**
>
> - *Oracle Database Licensing Information* for more information about Oracle Active Data Guard

10.2.1 Real-time Query

The `COMPATIBLE` database initialization parameter must be set to 11.0 or higher to use the real-time query feature of the Oracle Active Data Guard option.

A physical standby database instance cannot be opened if Redo Apply is active on a mounted instance of that database. 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 DISCONNECT;
```

> **Note:**
>
> If Redo Apply is active on an open instance, additional instances can be opened without having to stop Redo Apply.

Redo Apply cannot be started on a mounted physical standby instance if any instance of that database is open. The instance must be opened before starting Redo Apply on it.

Example: Querying V$DATABASE to Check the Standby's Open Mode

This example shows how the value of the `V$DATABASE.OPEN_MODE` column changes when a physical standby is open in real-time query mode.

1. Start up and open a physical standby instance, and perform the following SQL query to show that the database is open in read-only mode:

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

   OPEN_MODE
   --------------------
   READ ONLY
   ```

2. Issue the following SQL statement to start Redo Apply:

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

   Database altered.
   ```

3. Now that the standby is in real-time query mode (the standby is open in read-only mode and Redo Apply is active), the `V$DATABASE.OPEN_MODE` column changes to indicate the following:

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

   OPEN_MODE
   ```

```
--------------------
READ ONLY WITH APPLY
```

10.2.1.1 Monitoring Apply Lag in a Real-time Query Environment

If you are using real-time query to offload queries from a primary database to a physical standby database, you can monitor the apply lag to ensure that it is within acceptable limits.

The current apply lag is the difference, in elapsed time, between when the last applied change became visible on the standby and when that same change was first visible on the primary. This metric is computed to the nearest second.

To obtain the apply lag, query the `V$DATAGUARD_STATS` view. For example:

```
SQL> SELECT name, value, datum_time, time_computed FROM V$DATAGUARD_STATS -
> WHERE name like 'apply lag';

    NAME         VALUE            DATUM_TIME              TIME_COMPUTED
    ---------    -------------    -------------------     -------------------
    apply lag    +00 00:00:00     05/27/2009 08:54:16     05/27/2009 08:54:17
```

The `apply lag` metric is computed using data that is periodically received from the primary database. The `DATUM_TIME` column contains a timestamp of when this data was last received by the standby database. The `TIME_COMPUTED` column contains a timestamp taken when the `apply lag` metric was calculated. The difference between the values in these columns should be less than 30 seconds. If the difference is larger than this, the `apply lag` metric may not be accurate.

To obtain a histogram that shows the history of apply lag values since the standby instance was last started, query the `V$STANDBY_EVENT_HISTOGRAM` view. For example:

```
SQL> SELECT * FROM V$STANDBY_EVENT_HISTOGRAM WHERE NAME = 'apply lag' -
> AND COUNT > 0;

NAME             TIME       UNIT         COUNT        LAST_TIME_UPDATED
---------      ---------   --------    -----------    -----------------------
apply lag          0        seconds        79681          06/18/2009 10:05:00
apply lag          1        seconds         1006          06/18/2009 10:03:56
apply lag          2        seconds           96          06/18/2009 09:51:06
apply lag          3        seconds            4          06/18/2009 04:12:32
apply lag          4        seconds            1          06/17/2009 11:43:51
apply lag          5        seconds            1          06/17/2009 11:43:52

6 rows selected
```

To evaluate the apply lag over a time period, take a snapshot of `V$STANDBY_EVENT_HISTOGRAM` at the beginning of the time period and compare that snapshot with one taken at the end of the time period.

10.2.1.2 Configuring Apply Lag Tolerance in a Real-time Query Environment

The `STANDBY_MAX_DATA_DELAY` session parameter can be used to specify a session-specific apply lag tolerance, measured in seconds, for queries issued by non-administrative users to a physical standby database that is in real-time query mode.

This capability allows queries to be safely offloaded from the primary database to a physical standby database, because it is possible to detect if the standby database has become unacceptably stale.

If STANDBY_MAX_DATA_DELAY is set to the default value of NONE, than queries issued to a physical standby database are executed regardless of the apply lag on that database.

If STANDBY_MAX_DATA_DELAY is set to a nonzero value, then queries issued to a physical standby database are executed only if the apply lag is less than or equal to STANDBY_MAX_DATA_DELAY. Otherwise, an ORA-3172 error is returned to alert the client that the apply lag is too large.

If STANDBY_MAX_DATA_DELAY is set to 0, a query issued to a physical standby database is guaranteed to return the exact same result as if the query were issued on the primary database, unless the standby database is lagging behind the primary database, in which case an ORA-3172 error is returned.

Use the ALTER SESSION SQL statement to set STANDBY_MAX_DATA_DELAY. For example:

```
SQL> ALTER SESSION SET STANDBY_MAX_DATA_DELAY=2
```

10.2.1.3 Forcing Redo Apply Synchronization in a Real-time Query Environment

To ensure that all redo data received from the primary database has been applied to a physical standby database, you can use a SQL ALTER SESSION statement.

Issue the following SQL statement:

```
SQL> ALTER SESSION SYNC WITH PRIMARY;
```

This statement blocks until all redo data received by the standby database at the time that this command is issued has been applied to the physical standby database. An ORA-3173 error is returned immediately, and synchronization does not occur, if the redo transport status at the standby database is not SYNCHRONIZED or if Redo Apply is not active.

To ensure that Redo Apply synchronization occurs in specific cases, use the SYS_CONTEXT('USERENV','DATABASE_ROLE') function to create a standby-only trigger (enabled on the primary but that only takes certain actions if it is running on a standby). For example, you could create the following trigger that would execute the ALTER SESSION SYNC WITH PRIMARY statement for a specific user connection at logon:

```
CREATE TRIGGER adg_logon_sync_trigger
 AFTER LOGON ON user.schema
  begin
    if (SYS_CONTEXT('USERENV', 'DATABASE_ROLE')  IN ('PHYSICAL STANDBY')) then
      execute immediate 'alter session sync with primary';
    end if;
  end;
```

10.2.1.4 Real-time Query Restrictions

This list discusses restrictions related to real-time query mode.

- The apply lag control and Redo Apply synchronization mechanisms described above require that the client be connected and issuing queries to a physical standby database that is in real-time query mode.
- The following additional restrictions apply if STANDBY_MAX_DATA_DELAY is set to 0 or if the ALTER SESSION SYNC WITH PRIMARY SQL statement is used:
  - The standby database must receive redo data via the SYNC transport.

  - The redo transport status at the standby database must be SYNCHRONIZED and the primary database must be running in either maximum protection mode or maximum availability mode.
  - Real-time apply must be enabled.
- Oracle Active Data Guard achieves high performance of real-time queries in an Oracle RAC environment through the use of cache fusion. This allows the Oracle Data Guard apply instance and queries to work out of cache and not be slowed down by disk I/O limitations. A consequence of this is that an unexpected failure of the apply instance leaves buffers in inconsistent states across all the open Oracle RAC instances. To ensure data consistency and integrity, Oracle Data Guard closes all the other open instances in the Oracle RAC configuration, and brings them to a mounted state. You must manually reopen the instances - at which time the data is automatically made consistent, followed by restarting redo apply on one of the instances. In an Oracle Data Guard broker configuration, the instances are automatically reopened and redo apply is automatically restarted on one of the instances.

See Also:

- *Oracle Data Guard Broker* for more information about how the broker handles apply instance failures
- The My Oracle Support note 1357597.1 at `http://support.oracle.com` for additional information about apply instance failures in an Oracle Active Data Guard Oracle RAC standby

10.2.1.5 Automatic Block Media Recovery

If corrupt data blocks are encountered when a database is accessed, they can be automatically replaced with uncorrupted copies of those blocks.

This requires the following conditions:

- The physical standby database must be operating in real-time query mode, which requires an Oracle Active Data Guard license.
- The physical standby database must be running real-time apply.

Automatic block media recovery works in two directions depending on whether the corrupted blocks are encountered on the primary or on the standby.

Corrupted Blocks On the Primary

If corrupt data blocks are encountered at a primary database, then the primary automatically searches for good copies of those blocks on a standby and, if they are found, has them shipped back to the primary.

The primary requires a `LOG_ARCHIVE_DEST_`*n* to the standby only (a physical standby, a cascading physical standby, or a far sync instance). The primary does not require a `LOG_ARCHIVE_DEST_`*n* to any terminal destinations; it is able to automatically ascertain their service names.

Corrupted Blocks On a Standby

If corrupt data blocks are encountered at a standby, then the standby automatically initiates communication with the primary and requests uncorrupted copies of those blocks. For the primary to be able to ship the uncorrupted blocks to the standby, the following database initialization parameters must be configured on the standby. This is true even if the primary does not directly service the standby (for example, in cascading configurations).

- The `LOG_ARCHIVE_CONFIG` parameter is configured with a `DG_CONFIG` list and a `LOG_ARCHIVE_DEST_`*n* parameter is configured for the primary database.

  or

- The `FAL_SERVER` parameter is configured and its value contains an Oracle Net service name for the primary database.

Additional Automatic Block Media Repair Considerations

- Automatic repair is supported with any Oracle Data Guard protection mode. However, the effectiveness of repairing a corrupt block at the primary using the non-corrupt version of the block from the standby depends on how closely the standby apply is synchronized with the redo generated by the primary.
- When an automatic block repair has been performed, a message is written to the database alert log.
- If automatic block repair is not possible, an `ORA-1578` error is returned.

10.2.1.6 Manual Block Media Recovery

The RMAN `RECOVER BLOCK` command is used to manually repair a corrupted data block.

This command searches several locations for an uncorrupted copy of the data block. By default, one of the locations is any available physical standby database operating in real-time query mode. The `EXCLUDE STANDBY` option of the RMAN `RECOVER BLOCK` command can be used to exclude physical standby databases as a source for replacement blocks.

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

10.2.1.7 Tuning Queries on a Physical Standby Database

Queries on a physical standby database can be tuned for optimal performance.

For details about tuning queries, see the *Active Data Guard Best Practices* white paper available on the Oracle Maximum Availability Architecture (MAA) home page at:

`http://www.oracle.com/goto/maa`

Force Full Database Caching Mode

The use of force full database caching mode can potentially improve performance because queries are executed faster.

The enabling and disabling of force full database caching mode is not recorded in redo, so the status of in-memory caching is not necessarily the same on all members of a Data Guard configuration.

For more information about the Force Full Database In-Memory Caching feature, including guidelines on how and when to enable and disable it, see *Oracle Database Performance Tuning Guide*.

10.2.1.8 Adding Temp Files to a Physical Standby

If you are using a standby to offload queries from the primary database, then the standby must be configured with the minimum of one temp tablespace with at least one temporary data file.

If the nature of the workload requires more temp table space than is automatically created when the standby is first created, then you may need to manually add additional space.

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

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

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

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

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

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

10.2.2 DML Operations on Temporary Tables on Oracle Active Data Guard Instances

Redo generation on a read-only database is not allowed. When a data manipulation language (DML) operation makes a change to a global temporary table, the change itself does not generate redo since it is only a temporary table. However, the undo generated for the change does in turn generate redo. Prior to Oracle Database 12*c* Release 1 (12.1), this meant that global temporary tables could not be used on Oracle Active Data Guard standbys, which are read-only.

As of Oracle Database 12c Release 1 (12.1), the temporary undo feature allows the undo for changes to a global temporary table to be

stored in the temporary tablespace as opposed to the undo tablespace.

Undo stored in the temporary tablespace does not generate redo, thus enabling redo-less changes to global temporary tables. This allows DML operations on global temporary tables on Oracle Active Data Guard standbys.

This feature benefits Oracle Data Guard in the following ways:

Redo generation on a read-only database is not allowed. When a data manipulation language (DML) operation makes a change to a global temporary table, the change itself does not generate redo since it is only a temporary table. However, the undo generated for the change does in turn generate redo. Prior to Oracle Database 12*c* Release 1 (12.1), this meant that global temporary tables could not be used on Oracle Active Data Guard standbys, which are read-only.

However, as of Oracle Database 12c Release 1 (12.1), the temporary undo feature allows the undo for changes to a global temporary table to be stored in the temporary tablespace as opposed to the undo tablespace. Undo stored in the temporary tablespace does not generate redo, thus enabling redo-less changes to global temporary tables. This allows DML operations on global temporary tables on Oracle Active Data Guard standbys. This feature benefits Oracle Data Guard in the following ways:

- Read-mostly reporting applications that use global temporary tables for storing temporary data can be offloaded to an Oracle Active Data Guard instance.
- When temporary undo is enabled on the primary database, undo for changes to a global temporary table are not logged in the redo and thus, the primary database generates less redo. Therefore, the amount of redo that Oracle Data Guard must ship to the standby is also reduced, thereby reducing network bandwidth consumption and storage consumption.

To enable temporary undo on the primary database, use the `TEMP_UNDO_ENABLED` initialization parameter. On an Oracle Active Data Guard standby, temporary undo is always enabled by default so the `TEMP_UNDO_ENABLED` parameter has no effect.

> **Note:**
>
> Data definition language (DDL) operations on global temporary tables (for example, `CREATE` and `DROP`) must still be issued from the primary database. DDL changes are visible on the standby when it catches up with the primary database.

Restrictions

- The temporary undo feature requires that the database initialization parameter `COMPATIBLE` be set to 12.0.0 or higher.
- The temporary undo feature on Oracle Active Data Guard instances does not support temporary `BLOB`s or temporary `CLOB`s.
- Distributed transactions on an Oracle Active Data Guard instance are not permitted if they involve changes to local objects. For example, you cannot commit a transaction that modifies a global temporary table on the Oracle Active Data Guard instance and also updates a remote table on another database using a

database link. You must commit or roll back any outstanding DML operations to global temporary tables on the Active Data Guard instance before issuing a remote DML operation, or vice versa. This also includes implicit writes to global temporary tables made by operations such as `EXPLAIN PLAN` statements.

> **See Also:**
>
> - *Oracle Database Administrator's Guide* for more information about temporary undo
> - *Oracle Database Reference* for more information about the `TEMP_UNDO_ENABLED` initialization parameter

10.2.3 IM Column Store in an Active Data Guard Environment

As of Oracle Database 12*c* Release 2 (12.2.0.1), the Oracle Database In-Memory column store (IM column store) is supported on a standby database in an Active Data Guard (ADG) environment.

A reporting workload executing on an Active Data Guard standby database can use the IM column store. Using the IM column store improves the execution performance of the workload because it can take full advantage of accessing data in a compressed columnar format, in memory. Additionally, it is possible to populate a completely different set of data in the IM column store on the primary and standby databases, effectively doubling the size of the IM column store available to the application.

Note the following restrictions:

- In-Memory Expressions are captured based only on the queries executed on the primary database.
- In-Memory Information Lifecycle Management (ILM) polices based on access criteria are triggered based only on access recorded on the primary database.

> **Note:**
>
> - In-Memory FastStart is not supported on a standby database in an ADG environment.
> - In-Memory Join-Groups are not supported on a standby database in an ADG environment.
> - In-Memory column store is not supported with multi-instance redo apply in an ADG environment.

Related Topics:

- *Oracle Database In-Memory Guide*

10.2.4 Using Sequences in Oracle Active Data Guard

In an Oracle Active Data Guard environment, sequences created by the primary database with the default `CACHE` and `NOORDER` options can be accessed from standby databases as well.

When a standby database accesses such a sequence for the first time, it requests that the primary database allocate a range of sequence numbers. The range is based on the cache size and other sequence properties specified when the sequence was created. Then the primary database allocates those sequence numbers to the requesting standby database by adjusting the corresponding sequence entry in the data dictionary. When the standby has used all the numbers in the range, it requests another range of numbers.

The primary database ensures that each range request from a standby database gets a range of sequence numbers that do not overlap with the ones previously allocated for both the primary and standby databases. This generates a unique stream of sequence numbers across the entire Oracle Data Guard configuration.

Because the standby's requests for a range of sequences involve a round-trip to the primary, be sure to specify a large enough value for the `CACHE` keyword when you create a sequence to be used on an Oracle Active Data Guard standby. Otherwise, performance could suffer.

Also, be sure the terminal standby has a `LOG_ARCHIVE_DEST_`*n* parameter defined that points back to the primary.

Example: Assigning a Range of Sequence Values In a Multi-standby Configuration

This example shows how a range of sequence values can be assigned to a database when it references `NEXTVAL` on the sequence either for the first time or after it uses up all of the previously assigned sequence values. In this example, there are two standby databases.

1. On the primary database, issue the following SQL statements to create a global temporary table named `gtt`, and a sequence named `g` with a cache size of 10:

```
SQL> CREATE GLOBAL TEMPORARY TABLE gtt (a int);

Table created.

SQL> CREATE SEQUENCE g CACHE 10;

Sequence created.
```

2. On the first standby database, issue the following SQL statements:

```
SQL> INSERT INTO gtt VALUES (g.NEXTVAL);

1 row created.

SQL> INSERT INTO gtt VALUES (g.NEXTVAL);

1 row created.

SQL> SELECT * FROM gtt;

         A
```

```
----------
         1
         2
```

Because the sequence cache size was set to 10 (in Step 1) and because this is the first time the sequence was accessed, the results of the `SELECT` statement show that the first standby database is assigned sequence numbers 1 to 10.

3. On the primary database, issue the following SQL statements:

```
SQL> SELECT g.NEXTVAL FROM dual;

   NEXTVAL
----------
        11

SQL> SELECT g.NEXTVAL FROM dual;

   NEXTVAL
----------
        12
```

The results of the `SELECT` statements show that the primary database is assigned the next range of sequence values, 11-20.

4. On the second standby database, issue the following SQL statements:

```
SQL> INSERT INTO gtt VALUES (g.NEXTVAL);

1 row created.

SQL> INSERT INTO gtt VALUES (g.NEXTVAL);

1 row created.

SQL> SELECT * FROM gtt;

         A
----------
        21
        22
```

The results of the `SELECT` statement show that the second standby is assigned the next range of sequence values, 21-30.

> **Note:**
>
> Sequences created with the `ORDER` or `NOCACHE` options cannot be accessed on an Oracle Active Data Guard standby.

10.2.4.1 Session Sequences

A session sequence is a special type of sequence that is specifically designed to be used with global temporary tables that have session visibility.

Unlike the existing regular sequences (referred to as "global" sequences for the sake of comparison), a session sequence returns a unique range of sequence numbers only within a session, but not across sessions. Another difference is that session

sequences are not persistent. If a session goes away, so does the state of the session sequences that were accessed during the session.

Session sequences support most of the sequence properties that are specified when the sequence is defined. However, the CACHE/NOCACHE and ORDER/NOORDER options are not relevant to session sequences and are ignored.

Session sequences must be created by a read/write database but can be accessed on any read/write or read-only databases (either a regular database temporarily open read-only or a standby database).

Creating and Altering Session Sequences

To create a session sequence, issue the following SQL statement:

```
SQL> CREATE SEQUENCE … SESSION;
```

To alter an existing session sequence to be a regular ("global") sequence, issue the following SQL statement:

```
SQL> ALTER SEQUENCE … GLOBAL;
```

To alter a regular sequence to be a session sequence, issue the following SQL statement:

```
SQL> ALTER SEQUENCE … SESSION;
```

Example: Using Session Sequences

This example shows how session sequence values are unique to each database session.

1. On the primary database, issue the following SQL statements to create a global temporary table named gtt and a session sequence named s:

```
SQL> CREATE GLOBAL TEMPORARY TABLE gtt (a int);

Table created.

SQL> CREATE SEQUENCE s SESSION;

Sequence created.
```

2. On the standby database, issue the following SQL statements:

```
SQL> INSERT INTO gtt VALUES (s.NEXTVAL);

1 row created.

SQL> INSERT INTO gtt VALUES (s.NEXTVAL);

1 row created.


SQL> SELECT * FROM gtt;

         A
----------
         1
         2
```

3. From another session of the same standby database, issue the following SQL statements:

```
SQL> INSERT INTO gtt VALUES (s.NEXTVAL);

1 row created.

SQL> INSERT INTO gtt VALUES (s.NEXTVAL);

1 row created.


SQL> SELECT * FROM gtt;

         A
----------
         1
         2
```

The results of the `SELECT` statement show that the sequence values assigned are the same as those assigned for the first session in the previous step. This is because sequence values are unique to each database session.

10.3 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, but there are some changes that require manual intervention.

The following table lists primary database structural and configuration changes that require manual intervention at a physical standby database.

Table 10-1 Primary Database Changes That Require Manual Intervention at a Physical Standby

Primary Database Change	Action Required on Physical Standby Database
Adding a Data File or Creating 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 data file must be copied to the physical standby database.
Dropping Tablespaces and Deleting Data Files	Delete data file from primary and physical standby database after the redo data containing the `DROP` or `DELETE` command is applied to the physical standby.
Using Transportable Tablespaces with a Physical Standby Database	Move tablespace between the primary and the physical standby database.
Renaming a Data File in the Primary Database	Rename the data file on the physical standby database.
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.
NOLOGGING or Unrecoverable Operations	Use the RMAN command `RECOVER ... NONLOGGED BLOCK` to replace the invalid blocks on the standby with the changed blocks from the primary.

Table 10-1 (Cont.) Primary Database Changes That Require Manual Intervention at a Physical Standby

Primary Database Change	Action Required on Physical Standby Database
Refresh the Password File	As of Oracle Database 12*c* Release 2 (12.2.0.1), password file changes done on the primary database are automatically propagated to standby databases. The only exception to this is far sync instances. Updated password files must still be manually copied to far sync instances because far sync instances receive redo, but do not apply it. After the password file is up-to-date at the far sync instance, the redo containing the password update at the primary is automatically propagated to any standby databases that are set up to receive redo from that far sync instance. The password file is updated on the standby when the redo is applied.
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.
Initialization Parameters	Evaluate whether a corresponding change must be made to the initialization parameters on the physical standby database.

10.3.1 Adding a Data File or Creating a Tablespace

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

- If the `STANDBY_FILE_MANAGEMENT` database parameter on the physical standby database is set to `AUTO`, any new data files 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 data file must be manually copied from the primary database to the physical standby databases after it is added to the primary database.

> **Note:**
>
> On a physical standby for which the Oracle Active Data Guard option has been enabled, you cannot use the manual copy method. Instead, you must execute the following SQL statement on the standby to create an empty data file:
>
> ```
> SQL> ALTER DATABASE CREATE DATAFILE [filename | filenumber] -
> AS [NEW | new_filename];
> ```
>
> You must specify which one to rename: the `filename` or the `filenumber`.
>
> Also specify either the new filename or `NEW`. The `NEW` keyword lets Oracle automatically choose a name, if Oracle Managed Files (OMF) is enabled.

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

10.3.2 Dropping Tablespaces and Deleting Data Files

When a tablespace is dropped or a data file is deleted from a primary database, the corresponding data file(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 data files are no longer part of the database, query the `V$DATAFILE` view.

Delete the corresponding data file 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 data file for the tablespace. For example:

```
% rm /disk1/oracle/oradata/payroll/tbs_4.dbf
```

10.3.2.1 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 data files 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 tbs_4 INCLUDING CONTENTS AND DATAFILES;
SQL> ALTER SYSTEM SWITCH LOGFILE;
```

10.3.3 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 data files 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 data files and the export file to the primary database.

b. Copy the data files to the standby database.

The data files must have the same path name on the primary and standby databases unless the `DB_FILE_NAME_CONVERT` database initialization parameter has been configured. If `DB_FILE_NAME_CONVERT` has *not* been configured and the path names of the data files are not the same on the primary and standby databases, issue the `ALTER DATABASE RENAME FILE` statement to rename the data files. Do this after Redo Apply has failed to apply the redo generated by plugging the tablespace into the primary database. The `STANDBY_FILE_MANAGEMENT` initialization parameter must be set to `MANUAL` before renaming the data files, and then reset to the previous value after renaming the data files.

3. Plug in the tablespace.

   Invoke the Data Pump utility to plug the set of tablespaces into the primary database. Redo data is 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.*

10.3.4 Renaming a Data File in the Primary Database

When you rename one or more data files in the primary database, the change is not propagated to the standby database. It must be done manually.

To rename the same data files 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 data file in the primary database and manually propagate the changes to the standby database.

1. To rename the data file 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 data file on the primary system:

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

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

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

SQL> ALTER TABLESPACE tbs_4 ONLINE;
```

> **Note:**
>
> An alternative to these first three steps is to use the `ALTER DATABASE MOVE DATAFILE` command to rename a datafile. This command lets you rename a datafile while allowing read/write access to the datafile. Adequate storage area is a prerequisite for moving a datafile because during the execution of the `MOVE DATAFILE` command, the database maintains both the original and the renamed datafiles as two separate files. See Moving the Location of Online Data Files for more information.

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 data file 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 data file in the standby control file. To rename a data file, you must set the `STANDBY_FILE_MANAGEMENT` database initialization parameter to `MANUAL`. You can then reset the parameter to its previous value after renaming the data file.

```
SQL> ALTER DATABASE RENAME FILE '/disk1/oracle/oradata/payroll/tbs_4.dbf' -
> TO '/disk1/oracle/oradata/payroll/tbs_x.dbf';
```

9. On the standby database, restart Redo Apply:

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

If you do not rename the corresponding data file at the standby system, and then try to refresh the standby database control file, the standby database attempts to use the renamed data file, but will not find it. Error messages similar to the following are written to 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'
```

> **Note:**
>
> An alternative to steps 4-9 is to use the `ALTER DATABASE MOVE DATAFILE` command to rename a datafile at the standby. See Moving the Location of Online Data Files for more information.

10.3.5 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 log file group on the primary database.

Take the following steps to add or drop a log file group or standby 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.

> **Note:**
>
> An online logfile group must always be manually cleared before it can be dropped from a physical standby database. For example:
>
> ```
> ALTER DATABASE CLEAR LOGFILE GROUP 3;
> ```
>
> An online logfile group that has a status of `CURRENT` or `CLEARING_CURRENT` cannot be dropped from a physical standby database. An online logfile group that has this status can be dropped after a role transition.

4. Restore the `STANDBY_FILE_MANAGEMENT` initialization parameter and the Redo Apply options to their original states.
5. Restart Redo Apply.

In Oracle RAC environments, keep the following in mind:

- When an online redo log group is added to a primary database, you must manually add an online redo log group to the standby database. It is not done automatically.
- When a new redo thread is added to a primary database, a new redo thread is automatically added to the standby. By default, the new thread is configured with 2 log groups of 100 MB each. This cannot be changed or overridden.
- When a new log group is added to an existing redo thread, a new log group is not automatically added to its existing thread.

10.3.6 NOLOGGING or Unrecoverable Operations

When you perform a DML or DDL operation using the `NOLOGGING` or `UNRECOVERABLE` clause, blocks on the standby may be marked as invalid (also known as nonlogged blocks).

See Precedence of FORCE LOGGING Settings for details about when an operation is actually performed in a nonlogged fashion.

See Recovering After the NOLOGGING Clause Is Specified for information about recovering after the `NOLOGGING` clause is used.

See the *Oracle Database Administrator's Guide.* for information about specifying `FORCE LOGGING` mode.

10.3.7 Refresh the Password File

If the `REMOTE_LOGIN_PASSWORDFILE` database initialization parameter is set to `SHARED` or `EXCLUSIVE`, then the password file on a physical standby database is automatically replaced with a fresh copy from the primary database.

The file is replaced after administrative privileges are granted or revoked, or the password of a user with administrative privileges is changed. The only exception to this is far sync instances. Updated password files must still be manually copied to far sync instances because far sync instances receive redo, but do not apply it. When a password file is manually updated at a far sync instance, the redo containing the same password changes from the primary database is automatically propagated to any standby databases that are set up to receive redo from that far sync instance. The password file is updated on the standby when the redo is applied.

10.3.8 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 prevents access to encrypted columns on the physical standby database that are modified after the master encryption key is reset on the primary database.

For online tablespaces and databases, as of Oracle Database 12*c* Release 2 (12.2.0.1), you can encrypt, decrypt, and re-key both new and existing tablespaces, and existing databases within an Oracle Data Guard environment.

For offline tablespaces and databases, as of Oracle Database 12*c* Release 2 (12.2.0.1), you can encrypt and decrypt both new and existing tablespaces, and existing databases within an Oracle Data Guard environment.

In online conversion, the encryption, decryption, or re-keying on the standby is automatic after it is performed on the primary. An online encryption, decryption, or re-keying cannot be performed directly on a standby database.

In an offline conversion, the encryption or decryption must be performed manually on both the primary and standby. An offline conversion affects the data files on the particular primary or standby database only. Both the primary and physical standby should be kept at the same state. You can minimize downtime by encrypting (or decrypting) the tablespaces on the standby first, switching over to the primary, and then encrypting (or decrypting) the tablespaces on the primary.

See Also:

- *Oracle Database Advanced Security Guide*

10.4 Recovering Through the OPEN RESETLOGS Statement

Oracle 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) and the new redo branch from `OPEN RESETLOGS` has been registered at the standby	Redo Apply automatically takes the new branch of redo.	No manual intervention is necessary. The managed redo process (MRP) automatically resynchronizes the standby database with the new branch of redo data. Note: To check whether the new redo branch has been registered at the standby, perform the following query at the primary and standby and verify that the results match: `SELECT resetlogs_id, resetlogs_change# FROM V$DATABASE_INCARNATION WHERE status='CURRENT'`
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 Flashing Back a Physical Standby Database to a Specific Point-in-Time to flash back a physical standby database. 2. Restart Redo Apply to continue application of redo data onto new reset logs branch. The managed redo process (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 Creating a Physical Standby Database .
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.

10.5 Monitoring Primary, Physical Standby, and Snapshot Standby Databases

This table summarizes common primary database management actions and where to find information related to these actions.

Table 10-2 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$RECOVER_FILE` • `DBA_TABLESPACES`
Add or drop a data file or tablespace	• `DBA_DATA_FILES` • Alert log	• `V$DATAFILE` • Alert log
Rename a data file	• `V$DATAFILE` • Alert log	• `V$DATAFILE` • Alert log
Unlogged or unrecoverable operations	• `V$DATAFILE` • `V$DATABASE`	Alert log
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

Table 10-2 (Cont.) Sources of Information About Common Primary Database Management Actions

Primary Database Action	Primary Site Information	Standby Site Information
Change initialization parameter	Alert log	Alert log

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

You can 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$DATAGUARD_STATUS
- V$ARCHIVE_DEST

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

10.5.1.1 V$DATABASE

You can use the `V$DATABASE` view to display information about data protection, switchover status, and fast-start failover status.

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;
```

10.5.1.2 V$MANAGED_STANDBY

You can use the V$MANAGED_STANDBY view to query Redo Apply and redo transport status on a physical standby database.

For example, issue the following query:

```
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 remote file server (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.

10.5.1.3 V$ARCHIVED_LOG

You can use the V$ARCHIVED_LOG view to query information about archived redo log files that have been received by a physical or snapshot standby database from a primary database.

For example, issue the following query:

```
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         74795
```

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

10.5.1.4 V$LOG_HISTORY

You can use the V$LOG_HISTORY view to query archived log history information.

For example, issue the following query:

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

10.5.1.5 V$DATAGUARD_STATUS

You can use the V$DATAGUARD_STATUS view to display messages generated by Oracle Data Guard events that caused a message to be written to the alert log or to a server process trace file.

For example, issue the following query :

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

10.5.1.6 V$ARCHIVE_DEST

You can query the `V$ARCHIVE_DEST` view to show the status of each redo transport destination, and for redo transport destinations that are standby databases, the SCN of the last primary database redo applied at that standby database.

For example, issue the following query:

```
SQL> SELECT DEST_ID, APPLIED_SCN FROM V$ARCHIVE_DEST WHERE TARGET='STANDBY';

DEST_ID    STATUS    APPLIED_SCN
---------- --------- -----------
2          VALID     439054
3          VALID     439054
```

10.6 Tuning Redo Apply

Oracle white papers are available that describe how to optimize Redo Apply and media recovery performance.

In particular, see *Active Data Guard 11g Best Practices (includes best practices for Redo Apply*) . This paper is available on the Oracle Maximum Availability Architecture (MAA) home page at:

`http://www.oracle.com/goto/maa`

> **See Also:**
>
> My Oracle Support note 454848.1 at `http://support.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

10.7 Tuning Databases in an Active Data Guard Environment with SQL Tuning Advisor

In an Active Data Guard environment, SQL Tuning Advisor can tune a standby workload on a primary database.

Using database links, you can issue SQL Tuning Advisor statements on one database, but execute the statements on a different database.

Tuning a Standby Database Workload on a Primary Database

In some cases, a standby database can assume a reporting role in addition to its data protection role. The standby database can have its own workload of queries, some of which may require tuning. In this scenario, you tune a standby database workload by issuing every tuning statement on the standby database, but SQL Tuning Advisor performs its analysis on the primary database by using a standby-to-primary database link.

The following are the tasks that must be performed to tune a standby database workload on a primary database. The tasks must be performed at the standby database in the order given, using the `DBMS_SQLTUNE` PL/SQL package:

1. Execute the `DBMS_SQLTUNE.CREATE_TUNING_TASK` statement to fetch the data from the primary database needed to create a task. Because the standby is a read-only database, the data about the task is written remotely to the primary database. A database link parameter is required in this step to write to the primary. (A database link parameter is optional in subsequent steps, because it is tied to the task created in this step.)
2. Execute the `DBMS_SQLTUNE.EXECUTE_TUNING_TASK` statement. Initially, the data required to execute a task is fetched from the remote primary database. The tuning analysis process to find the possible recommendations is executed. Because the standby is a read-only database, when the results are available they are stored remotely at the primary.
3. Execute the `DBMS_SQLTUNE.REPORT_TUNING_TASK` statement. The data needed to construct a report is stored remotely at the primary database. The data is fetched remotely from the primary and constructed locally at the standby.
4. Execute the `DBMS_SQLTUNE.ACCEPT_SQL_PROFILE` statement. The profile data is written to the remote primary database because the standby is read-only.
5. The SQL Profiles are made available at the standby using Redo Apply.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for more information about the `DBMS_SQLTUNE` package
- *Oracle Database SQL Tuning Guide*for more information about local and remote SQL tuning

10.8 Using Oracle Diagnostic Pack to Tune Oracle Active Data Guard Standbys

The Oracle Diagnostic Pack can be used with an Oracle Active Data Guard standby database that is open in read-only mode.

This enables you to capture performance data to the Automatic Workload Repository (AWR) for an Oracle Active Data Guard standby and to run Automatic Database Diagnostic Monitor (ADDM) analysis on the AWR data. For details about how to perform these operations, see *Oracle Database Performance Tuning Guide*.

10.9 Managing a Snapshot Standby Database

A snapshot standby database is a fully updatable standby database. It receives and archives redo data from a primary database, but does not apply it.

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 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 is then 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 increased time to recover from primary database failures.

10.9.1 Converting a Physical Standby Database into a Snapshot Standby Database

These steps describe how to convert a physical standby database into a snapshot standby database.

1. Stop Redo Apply, if it is active.
2. Ensure that the database is mounted, but not open.
3. Ensure that a fast recovery area has been configured. It is not necessary for flashback database to be enabled.
4. Issue the following SQL statement to perform the conversion:

   ```
   SQL> ALTER DATABASE CONVERT TO SNAPSHOT STANDBY;
   ```

5. Open the snapshot standby in read/write mode by issuing the following SQL statement:

   ```
   SQL> ALTER DATABASE OPEN READ WRITE;
   ```

> **Note:**
>
> A physical standby database that is managed by the Oracle Data Guard broker can be converted into a snapshot standby database using either DGMGRL or Oracle Enterprise Manager Cloud Control. See *Oracle Data Guard Broker* for more details.

10.9.2 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 Oracle 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 prevents a snapshot standby from being converted back to a physical standby.
>
> For information about some of the limitations of Flashback Database, see *Oracle Database Backup and Recovery User's Guide*.

10.9.3 Converting a Snapshot Standby Database into a Physical Standby Database

These steps describe how to convert a snapshot standby database into a physical standby database.

1. On an Oracle Real Applications Cluster (Oracle 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;
```

Redo data received while the database was a snapshot standby database is 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.

11
Managing a Logical Standby Database

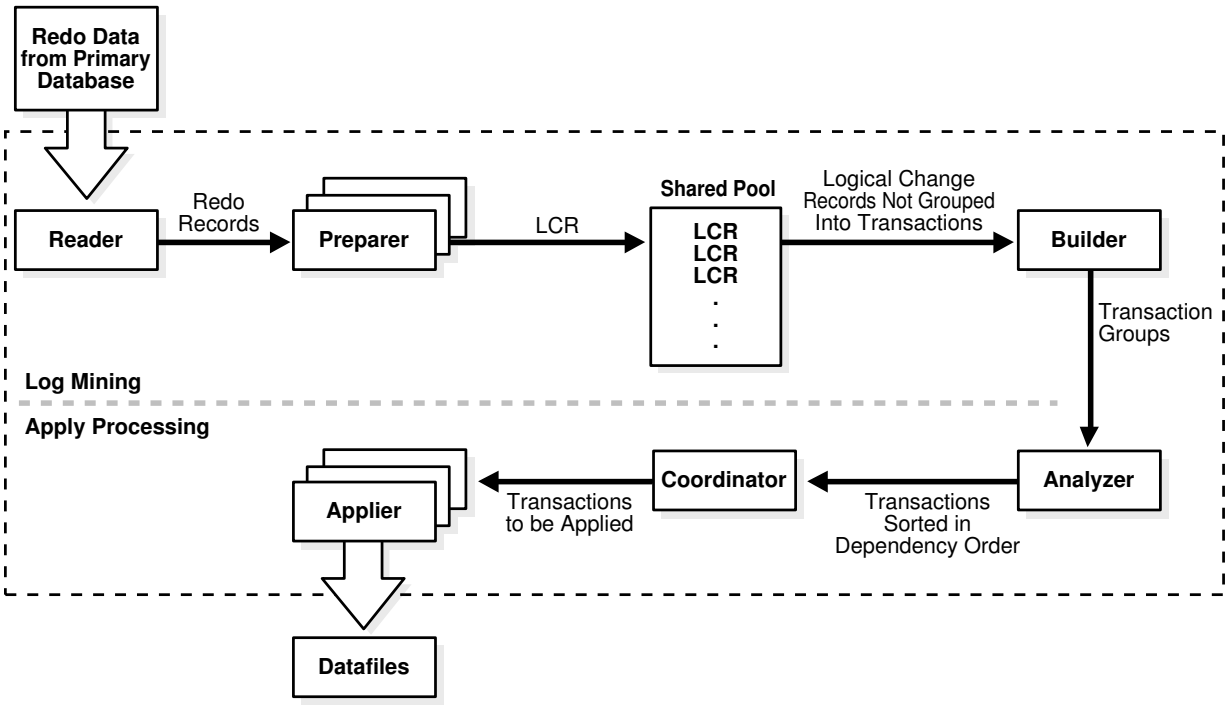
An understanding of these concepts will help you to successfully manage a logical standby database.

11.1 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 11-1 shows the flow of information and the role that each process performs.

Figure 11-1 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 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.

11.1.1 Various Considerations for SQL Apply

Understanding these concepts about transaction size, pageouts, restarts, DML Apply, and password verification will help you to manage logical standbys to their best advantage.

See the following:

- Transaction Size Considerations
- Pageout Considerations
- Restart Considerations
- DML Apply Considerations
- DDL Apply Considerations
- Password Verification Functions

11.1.1.1 Transaction Size Considerations

SQL Apply categorizes transactions into two classes: small and large.

Definitions of each class are as follows:

- 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 eight million rows. For transactions larger than 8 million rows, SQL Apply uses the temporary segment to stage some of the internal metadata required to process the transaction. Be sure to allocate enough space in your temporary segment for SQL Apply to successfully process transactions larger than 8 million rows.

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.

11.1.1.2 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 pages 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 occur occasionally and should not effect the overall throughput of the system.

> **See Also:**
>
> See Customizing a Logical Standby Database for more information about how to identify problematic pageouts and perform corrective actions

11.1.1.3 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, prevent the `RESTART_SCN` from advancing for the duration of the workload.

11.1.1.4 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 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.

11.1.1.5 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 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 reissues the same DDL at the logical standby, the function `sysdate()` is reevaluated at the logical standby. Thus, the column `start_date` is 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 are not 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 is 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 stop SQL Apply because the `MODIFY_DATE` column data recorded in the redo stream does not match the data that exists at the logical standby database.

11.1.1.6 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 are not 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.

11.2 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` to use 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 Modifying a Logical Standby Database.

> **Note:**
>
> Do not let the primary and logical standby databases diverge while the database guard is disabled.

11.3 Views Related to Managing and Monitoring a Logical Standby Database

You can use performance views to monitor the behavior of SQL Apply maintaining a logical standby database.

The following topics 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

11.3.1 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 stops 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 -
> 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.

11.3.2 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.

11.3.3 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.

11.3.4 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:**
>
> Monitoring SQL Apply Progress for example output

11.3.5 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:**
>
> Monitoring SQL Apply Progress for example output

11.3.6 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:**
>
> Monitoring SQL Apply Progress for example output

11.3.7 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.
```

11.4 Monitoring a Logical Standby Database

When working with logical standby databases, you can monitor SQL Apply progress, and also the automatic deletion of log files.

See the following topics:

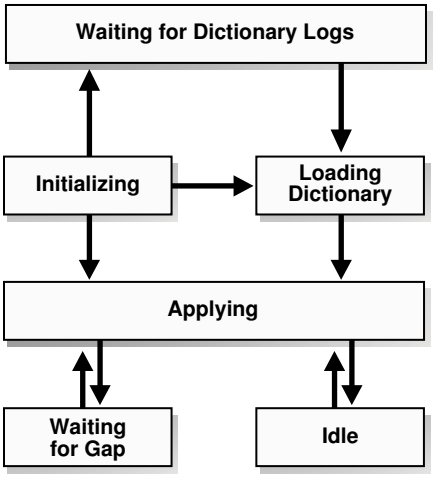
- Monitoring SQL Apply Progress
- Automatic Deletion of Log Files

11.4.1 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 11-2 shows the flow of these states.

Figure 11-2 Progress States During SQL Apply Processing



The following subsections describe each state in more detail.

Initializing State

When you start SQL Apply by issuing an `ALTER DATABASE START LOGICAL STANDBY APPLY` statement, it goes into 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 stays 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 do 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) is always 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-16240: waiting for log file (thread# 1, sequence# 99)
```

Idle State

SQL Apply enters this state once it has applied all redo generated by the primary database.

11.4.2 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 fast recovery area
- In a directory outside of the fast recovery area

Foreign archived logs stored in the fast 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 fast recovery area.

Foreign archived logs that are not stored in fast recovery area are by default managed by SQL Apply. Under automatic management, foreign archived logs that are not stored in the fast 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 fast 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.

11.5 Customizing a Logical Standby Database

A logical standby database can be customized in several ways, including logging of events, preventing changes to specific schema objects, and adding or re-creating tables.

See 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*

11.5.1 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 are not supported by a logical standby database, issue the following statements:

```
SQL> ALTER DATABASE STOP LOGICAL STANDBY APPLY;SQL> EXEC
DBMS_LOGSTDBY.APPLY_SET('RECORD_UNSUPPORTED_OPERATIONS', 'TRUE');SQL> ALTER DATABASE
START LOGICAL STANDBY APPLY IMMEDIATE;
```

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. Oracle Data Guard logs at least one unsupported operation per table in the DBA_LOGSTDBY_EVENTS view. Using SQL Apply to Upgrade the Oracle Database provides detailed information about rolling upgrades.

11.5.2 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;
```

11.5.3 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;
```

11.5.4 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.

11.5.4.1 Performing DDL on a Logical Standby Database

You can 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 SYSDG, SYSTEM, or another privileged account, you cannot 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 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 IMMEDIATE;
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. Doing so introduces deviations between the primary and standby databases that make it impossible for the logical standby database to be maintained.

11.5.4.2 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 writes to `TESTEMP%` in `HR`. The tables are no longer 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 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 SYSTEM.LOGSTDBY$PARAMETERS WHERE NAME = 'GUARD_STANDBY';

VALUE
---------------
Ready
```

When 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:**
>
> DDL Statements Supported by a Logical Standby Database and the `DBMS_LOGSTDBY` package in *Oracle Database PL/SQL Packages and Types Reference*

11.5.5 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 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', -
> table_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.

11.6 Managing Specific Workloads In the Context of a Logical Standby Database

You can manage specific workloads in the context of a logical standby.

See the following:

- Importing a Transportable Tablespace to the Primary Database
- Using Materialized Views
- How Triggers and Constraints Are Handled on a Logical Standby Database
- Using Triggers to Replicate Unsupported Tables
- Recovering Through the Point-in-Time Recovery Performed at the Primary
- Running an Oracle Streams Capture Process on a Logical Standby Database

11.6.1 Importing a Transportable Tablespace to the Primary Database

A transportable tablespace can be imported to a primary database.

Perform the following steps:

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.

11.6.2 Using Materialized Views

Logical standby automatically skips DDL statements related to materialized views.

For example, logical standby skips the following statements:

- 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 are not created on the logical standby database. However, materialized views created on the primary database prior to the logical standby database being created are 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. 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 (connect as user SYS before issuing these statements):

```
SQL> ALTER SESSION DISABLE GUARD;

SQL> CREATE MATERIALIZED VIEW LOG ON SCOTT.EMP -
>  WITH ROWID (EMPNO, ENAME, MGR, DEPTNO) INCLUDING NEW VALUES;

SQL> CREATE MATERIALIZED VIEW LOG ON SCOTT.DEPT -
>  WITH ROWID (DEPTNO, DNAME) INCLUDING NEW VALUES;

SQL> CREATE MATERIALIZED VIEW SCOTT.MANAGED_BY -
>  REFRESH ON DEMAND -
>  ENABLE QUERY REWRITE -
>  AS SELECT  E.ENAME, M.ENAME AS MANAGER -
>  FROM SCOTT.EMP E, SCOTT.EMP M WHERE E.MGR=M.EMPNO;

SQL> CREATE MATERIALIZED VIEW SCOTT.IN_DEPT -
>  REFRESH FAST ON COMMIT -
>  ENABLE QUERY REWRITE -
>  AS SELECT E.ROWID AS ERID, D.ROWID AS DRID, E.ENAME, D.DNAME -
>  FROM SCOTT.EMP E, SCOTT.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.)

11.6.3 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.

For triggers and constraints on tables *not maintained* by SQL Apply:

- Constraints are evaluated
- Triggers are fired

11.6.4 Using Triggers to Replicate Unsupported Tables

DML triggers created on a table have their `DBMS_DDL.SET_TRIGGER_FIRING_PROPERTY fire_once` parameter set to `TRUE` by default.

The triggers fire only when the table is modified by a user process. They are automatically disabled inside SQL Apply processes, and thus do not fire when a SQL Apply process modifies the table. There are two ways to fire a trigger as a result of SQL Apply process making a change to a maintained table:

- Set the `fire_once` parameter of a trigger to `FALSE`, which allows it to fire in either the context of a user process or a SQL Apply process
- Set the `apply_server_only` parameter to `TRUE` which results in the trigger firing only in the context of a SQL Apply process and not in the context of a user process

fire_once	apply_server_only	description
TRUE	FALSE	This is the default property setting for a DML trigger. The trigger fires only when a user process modifies the base table.
FALSE	FALSE	The trigger fires in the context of a user process and in the context of a SQL Apply process modifying the base table. You can distinguish the two contexts by using the `DBMS_LOGSTDBY.IS_APPLY_SERVER` function.
TRUE/FALSE	TRUE	The trigger only fires when a SQL Apply process modifies the base table. The trigger does not fire when a user process modifies the base table. Thus, the `apply_server_only` property overrides the `fire_once` parameter of a trigger.

Tables that are unsupported due to simple object type columns can be replicated by creating triggers that fire in the context of a SQL Apply process (either by setting the `fire_once` parameter of such a trigger to `FALSE` or by setting the apply_server_only parameter of such a trigger to `TRUE`). A regular DML trigger can be used on the primary database to flatten the object type into a table that can be supported. The trigger that

fires in the context of a SQL Apply process on the logical standby reconstitutes the object type and updates 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 `VARRAY`s 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

--
-- Option#1 (Better Option: Create a trigger and
-- set its apply-server-only property to TRUE)
```

```
-- 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_aso
  after insert on employees_transfer for each row
begin

    insert into employees (IdNumber, Department, Info)
    values (:new.t_IdNumber, :new.t_Department,
Person(:new.t_FirstName, :new.t_LastName,  :new.t_BirthDate));

end

-- set this trigger to fire from the apply server
execute dbms_ddl.set_trigger_firing_property( -
trig_owner => 'scott', -
trig_name  => 'reconstruct_employees_aso',
property => dbms_ddl.apply_server_only,
setting => TRUE);

--
-- Option#2 (Create a trigger and set
--           its fire-once property to FALSE)
-- create this trigger at the logical standby database
-- to populate object table on the standby
-- this trigger will fire when apply replicates rows to -- the standby, but we will
need to make sure we are
-- are executing inside a SQL Apply process by invoking
-- dbms_logstdby.is_apply_server function
--
create or replace trigger reconstruct_employees_nfo
  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_nfo',
property => dbms_ddl.fire_once,
setting => FALSE);
```

11.6.5 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 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 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.

11.6.6 Running an Oracle Streams Capture Process on a Logical Standby Database

You can run an Oracle Streams capture process on a logical standby database to capture changes from any table that exists on the logical standby database (whether it is a local table or a maintained table that is being replicated from the primary database).

When changes are captured to a maintained table, there is additional latency as compared to running an Oracle Streams capture process at the primary database. The additional latency is because of the fact that when you are running at a logical standby, the Oracle Streams capture process must wait for the changes to be shipped from the primary to the logical standby and applied by SQL Apply. In most cases, if you are running real time apply, it is no more than a few seconds.

The Oracle Streams capture process is associated with the database where it was created; the role of the database is irrelevant. For example, suppose you have a primary database named `Boston` and a logical standby named `London`. You cannot move the Oracle Streams capture process from one database to the other as you go through role transitions. For instance, if you created an Oracle Streams capture process on `London` when it was a logical standby, then it remains on `London` even when `London` becomes the primary as a result of a role transition operation such as a

switchover or failover. For the Oracle Streams capture process to continue working after a role transition, you must write a role transition trigger such as the following:

```
create or replace trigger streams_aq_job_role_change1
after DB_ROLE_CHANGE on database
declare
cursor capture_aq_jobs is
  select job_name, database_role
   from dba_scheduler_job_roles
   where job_name like 'AQ_JOB%';
u capture_aq_jobs%ROWTYPE;
my_db_role  varchar2(16);
begin

  if (dbms_logstdby.db_is_logstdby() = 1) then my_db_role := 'LOGICAL STANDBY';
  else my_db_role := 'PRIMARY';
  end if;

 open capture_aq_jobs;
 loop
   fetch capture_aq_jobs into u;
   exit when capture_aq_jobs%NOTFOUND;

   if (u.database_role != my_db_role) then
     dbms_scheduler.set_attribute(u.job_name,
              'database_role',
               my_db_role);

   end if;
 end loop;
 close capture_aq_jobs;

exception
 when others then
 begin
   raise;
 end;
end;
```

11.7 Using Extended Datatype Support During Replication

Extended Datatype Support (EDS) provides a mechanism for logical standbys to support certain data types that lack native redo-based support.

For example, a table with a top-level `VARRAY` column can be replicated using EDS.

You can query the `DBA_LOGSTDBY_EDS_SUPPORTED` view to find out which tables are candidates for EDS.

EDS-based replication works seamlessly with role transitions. For example, if EDS-based replication is enabled between a primary database and a target logical standby database for a table named `OE.CUSTOMERS`, then after a switchover or failover operation, the `OE.CUSTOMERS` table continues to be replicated using the EDS framework. This is also true when a bystander standby is present and using EDS while replicating a table.

11.7.1 How EDS-Based Replication Works

The `DBMS_LOGSTDBY` PL/SQL package provides procedures that add, remove, or change extended datatype support (EDS).

EDS-based replication works through the use of triggers, a shadow table, and in some cases a materialized view.

The shadow table is created in the same tablespace as the target table. It contains the data from the base table that has been transformed into a format natively supported by logical standby. A shadow table is not partitioned, even if the target table is. A shadow table resides in the same partition as the base table. A shadow table contains data only through the duration of the transaction that is modifying the source table. As a result, shadow table size is minimal and does not depend on source table size.

A DML trigger is created on the base table and on the shadow table. The triggers are created in the `SYS` schema, which owns them.

The first trigger (the base trigger) fires whenever a DML operation (`INSERT`, `UPDATE`, or `DELETE`) takes place. The trigger transforms unsupported data types to logical standby supported data types and then captures the transformed row in a shadow table along with information about the type of DML operation.

The shadow table contains only logical standby supported data types; therefore it is replicated natively. A second trigger (the shadow trigger) fires for any DML operation done to the shadow table by an apply process on the logical standby. The shadow trigger transforms the data back to its original form and applies it to the target table according to whatever DML operation was recorded with the row.

> **Note:**
>
> You cannot use SQL*Loader direct path loads for EDS tables. The triggers on EDS tables cause the load to fail. Use conventional path instead.

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* for more information about EDS-related procedures provided in the `DBMS_LOGSTDBY` PL/SQL package

11.7.2 Enabling EDS-Based Replication At a Logical Standby

These steps provide an example of how to enable EDS-based replication.

The procedure is split between the primary and standby databases. Be sure you are on the correct database for each step.

1. On the primary database, identify which tables are candidates for EDS support by querying the `DBA_LOGSTDBY_EDS_SUPPORTED` view as follows:

   ```
   SQL> SELECT * FROM DBA_LOGSTDBY_EDS_SUPPORTED;
   ```

2. On the primary database, issue calls to the `DBMS_LOGSTDBY.EDS_ADD_TABLE` procedure for any table names returned from the query you just made. For example, suppose there is a table named `OE.CUSTOMERS` and it has an `SDO_GEOMETRY` column which excludes the table from native log-based replication. Execute the following command which would create triggers and a shadow table:

```
SQL> EXECUTE DBMS_LOGSTDBY.EDS_ADD_TABLE(table_owner =>'OE',
table_name =>'CUSTOMERS');
```

The primary database starts generating extra information in the redo stream.

3. On the standby database, stop SQL Apply:

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

4. On the standby database, issue the same set of calls as in Step 2 to `DBMS_LOGSTDBY.EDS_ADD_TABLE`, but add a database link to the primary:

```
SQL> EXECUTE DBMS_LOGSTDBY.EDS_ADD_TABLE(table_owner =>'OE',
table_name =>'CUSTOMERS',p_dblink => 'dblink_to_primary');
```

This statement enables EDS-based replication for the source table at the logical standby and also imports the data associated with the base table (along with all secondary objects such as indexes and constraints) to the standby.

5. On the standby database, restart SQL Apply:

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

At this point, the `OE.CUSTOMERS` table is being replicated by the EDS facility.

If there is more than one logical standby in the configuration, then Steps 3 - 5 must be repeated on each one.

11.7.3 Removing EDS-Based Replication From a Logical Standby

Use the `DBMS_LOGSTDBY.EDS_REMOVE_TABLE` procedure to remove EDS-based replication for a particular table at a logical standby.

If invoked on a standby, then this procedure removes support from only that standby.

If invoked on the primary, then this procedure removes support for that table on the primary and on all standbys. It also drops all metadata related to EDS-based replication at the primary database. Metadata related to EDS-based replication at logical standby databases is dropped after redo associated with the `EDS_REMOVE_TABLE` procedure is processed. Only EDS-based metadata and supporting objects are dropped; the source table remains unchanged.

You may want to remove EDS-based replication for a set of tables if you only added the support for the duration of a rolling upgrade.

11.7.4 How EDS-Based Replication Handles Skip Rules

EDS-based replication assumes that there are no DDL skip rules associated with the source table. It is expected that DDLs done on the source tables at the primary database are replicated using the native redo-based mechanism.

Additionally, the presence of a DML skip rule on a source table that is being replicated using EDS does not affect EDS-based replication.

Attempts to create EDS-based replication on a table that matches a wild card skip rule on a schema fail, as do attempts to add a wild card skip rule that matches an existing message table created for EDS.

11.7.5 How EDS-Based Replication Handles DDL

Because EDS-based replication relies on triggers that are generated according to a table's definition, it is possible that DDL operations might alter the table in such a way that the triggers are no longer valid.

The triggers must be regenerated, either automatically or manually, according to the table's new definition.

To enable (or disable) automatic DDL handling, you must use the `DBMS_LOGSTDBY.EDS_EVOLVE_AUTOMATIC` procedure. When automatic DDL handling is enabled, an EDS-specific DDL trigger fires after every DDL operation to determine whether the DDL in question affects the viability of EDS-based replication of any of the tables currently enabled for EDS. If it does, then a separate EVOLVING trigger is generated on the table which ensures that no DML operations are allowed on the affected base table until compensating actions are taken to repair the EDS-specific metadata. Once the compensating actions are taken, the EVOLVING trigger is dropped, allowing DDL operations on the table to resume.

Automatic DDL handling in the presence of EDS-based replication requires a DDL trigger that fires on all DDL operations. In some situations it may be useful to take compensating actions manually. In such cases, you can disable automatic DDL handling and use the `DBMS_LOGSTDBY.EDS_EVOLVE_MANUAL` procedure to handle DDL.

11.7.5.1 Enabling and Disabling Automatic DDL Handling

Use the `DBMS_LOGSTDBY.EDS_EVOLVE_AUTOMATIC` procedure to enable and disable automatic DDL handling on EDS-maintained tables.

To enable automatic DDL handling on all EDS-maintained tables, issue the following statement once at the primary database, prior to the first call to `DBMS_LOGSTDBY.EDS_ADD_TABLE`:

```
EXECUTE DBMS_LOGSTDBY.EDS_EVOLVE_AUTOMATIC('ENABLE');
```

To disable automatic DDL handling on all EDS-maintained tables, issue the following statement:

```
EXECUTE DBMS_LOGSTDBY.EDS_EVOLVE_AUTOMATIC('DISABLE');
```

11.7.5.2 Manually Handling DDL

To manually handle DDL operations, you must call the `DBMS_LOGSTDBY.EDS_EVOLVE_MANUAL` procedure before and after a DDL operation that may affect the base table being replicated with extended datatype support (EDS).

Take the following steps to handle DDL operations manually:

1. Call the `EDS_EVOLVE_MANUAL` procedure with the `START` option:

   ```
   EXECUTE DBMS_LOGSTDBY.EDS_EVOLVE_MANUAL('START');
   ```

2. Perform the DDL operation.
3. Call the `EDS_EVOLVE_MANUAL` procedure with the `FINISH` option:

   ```
   EXECUTE DBMS_LOGSTDBY.EDS_EVOLVE_MANUAL('FINISH');
   ```

Deviating from this order could result in errors and possible data loss.

If necessary, you can cancel manual DDL handling by invoking the procedure with the `CANCEL` option:

```
EXECUTE DBMS_LOGSTDBY.EDS_EVOLVE_MANUAL('CANCEL');
```

> **See Also:**
>
> - *Oracle Database PL/SQL Packages and Types Reference* for more information about EDS-related procedures in the `DBMS_LOGSTDBY` package

11.8 Tuning a Logical Standby Database

These topics provide information about various ways to tune logical standby databases.

- 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

11.8.1 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 -
> WHERE OWNER NOT IN (SELECT OWNER FROM DBA_LOGSTDBY_SKIP -
> WHERE STATEMENT_OPT = 'INTERNAL SCHEMA') -
> MINUS -
> SELECT DISTINCT TABLE_OWNER, TABLE_NAME FROM DBA_INDEXES -
> WHERE INDEX_TYPE NOT LIKE ('FUNCTION-BASED%') -
> MINUS -
> 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 ensures that the EMPNO column, which can be used to uniquely identify the rows in HR.TEST_EMPLOYEES table, is 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 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;
```

11.8.2 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:**
>
> - Ensure Table Rows in the Primary Database Can Be Uniquely Identified

11.8.3 Adjust the Number of 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.

The following sections describe:

- Adjusting the Number of APPLIER Processes
- Adjusting the Number of PREPARER Processes

11.8.3.1 Adjusting the Number of APPLIER Processes

Before adjusting the number of APPLIER processes, you should determine whether doing so will help you achieve greater throughput.

To determine this, perform the following steps:

1. Check whether 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 = 16116;

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:

```
SELECT NAME, VALUE FROM V$LOGSTDBY_STATS WHERE NAME = 'txns applied' OR NAME =
'distinct txns in queue';
```

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 (`distinct txns in queue - txns 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 prevents additional available APPLIER processes from applying them. For instance, if the majority of the transactions that are ready to be applied are DDL transactions, then adding more APPLIER processes does 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 must satisfy the following equation:

```
APPLY_SERVERS + PREPARE_SERVERS = MAX_SERVERS - 3
```

you must first set MAX_SERVERS to 24. Once you have done that, you can 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);
```

11.8.3.2 Adjusting the Number of PREPARER Processes

It is rare that you will need to adjust the number of PREPARER processes. Before increasing their number, you must ensure that certain conditions are true.

The conditions that must be true are as follows:

- 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 = 16116;

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 = 'txns applied' OR - >
NAME = 'distinct txns in queue';
```

```
NAME                          VALUE
---------------------         -------
txns applied                   27892
distinct txns in queue         12896

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 = 16116;

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 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);
```

11.8.4 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. Increasing memory allocated to the LCR cache may help.

To determine whether increasing memory allocated to the LCR cache would 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 (seconds)             894856
bytes paged out                           20000
pageout time (seconds)                        2
system idle time (seconds)                 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 (seconds)             895156
bytes paged out                         1020000
pageout time (seconds)                      100
system idle time (seconds)                 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
```

11.8.5 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 allows 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
```

You can change back 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.

11.9 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, then 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.

12

Using RMAN to Back Up and Restore Files

You can create backup strategies using Oracle Recovery Manager (RMAN) with Oracle Data Guard and standby databases.

RMAN can perform backups with minimal effect on the primary database and quickly recover from the loss of individual data files, or the entire database. RMAN and Oracle Data Guard can be used together to simplify the administration of an Oracle Data Guard configuration.

See 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 an Oracle Data Guard environment
> - *Oracle Database Backup and Recovery Reference* for detailed information about RMAN commands

12.1 About RMAN File Management in an Oracle Data Guard Configuration

RMAN uses a recovery catalog to track filenames for all database files in an Oracle 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.

12.1.1 Interchangeability of Backups in an Oracle Data Guard Environment

RMAN commands use the recovery catalog metadata to behave transparently across different physical databases in the Oracle 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 an Oracle Data Guard environment. RMAN automatically updates the filenames for database files during restore and recovery at the databases.

12.1.2 Association of Backups in an Oracle Data Guard Environment

The recovery catalog tracks the files in the Oracle 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.

12.1.3 Accessibility of Backups in an Oracle Data Guard Environment

In an Oracle 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 data file 1 on `prod` to `/prmhost/disk1/df1.dbf` on the production host and also to tape. RMAN backs up data file 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.

12.2 About RMAN Configuration in an Oracle Data Guard Environment

In an Oracle Data Guard configuration, the process of backing up control files, data files, 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 an Oracle 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;
```

12.3 Recommended RMAN and Oracle Database Configurations

These configurations 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

These configurations make the following assumptions:

- The standby database is a physical standby database, and backups are taken only on the standby database. See Standby Databases Too Geographically Distant to Share Backups 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 has 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 an Oracle 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) or later.

- Oracle Secure Backup software or 3rd-party media management software is configured with RMAN to make backups to tape.

12.3.1 Oracle Database Configurations on Primary and Standby Databases

These Oracle Database configurations are recommended on every primary and standby database in the Oracle Data Guard environment.

- Configure a fast recovery area for each database (the recovery area is local to a database).

  The fast recovery area is a single storage location on a file system or Oracle Automatic Storage Management (Oracle 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 fast 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 fast 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 fast recovery area:

  ```
  DB_RECOVERY_FILE_DEST = <mount point or Oracle 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 fast 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 fast 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

12.3.2 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 Oracle 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, to delete logs after ensuring that they *shipped* to all destinations, use the following configuration:

   ```
   CONFIGURE ARCHIVELOG DELETION POLICY TO SHIPPED TO ALL STANDBY;
   ```

   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` (or `SYSBACKUP`) 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

12.3.3 RMAN Configurations at a Standby Database Where Backups are Performed

These 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 data files 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. Because the archived logs are backed up at the standby database, Oracle recommends that you configure the `BACKED UP` option for the log deletion policy:

   ```
   CONFIGURE ARCHIVELOG DELETION POLICY BACKED UP n TIMES TO DEVICE TYPE SBT;
   ```

> **See Also:**
>
> *Oracle Database Backup and Recovery User's Guide* for more information about enabling deletion policies for archived redo logs

12.3.4 RMAN Configurations at a Standby Where Backups Are Not Performed

These 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 (this is also applicable to all terminal databases when the cascading or far sync instance features are in use):

```
CONFIGURE ARCHIVELOG DELETION POLICY TO APPLIED ON ALL STANDBY;
```

12.4 Backup Procedures

You can use RMAN scripts and procedures to back up Oracle Database in an Oracle Data Guard configuration.

See the following topics:

- 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

All backup operations can be offloaded to a single standby database, except backups of the SPFILE. Backups of the SPFILE can only be restored to the database from which they were backed up.

For databases that are not backed up, Oracle recommends that you at least back up the SPFILE to a known local disk location. If the SPFILE backups need to be further backed up to tape, you can copy them to the database site where backups to tape have been configured. The SPFILE backups can then be cataloged at that database using the following RMAN command:

```
CATALOG START WITH '<SPFILE backup directory>';
```

Then back up the SPFILE backups to tape:

```
BACKUP BACKUPSET ALL;
```

When the SPFILE needs to be restored for a specific database, the appropriate SPFILE backup is restored from disk or tape.

12.4.1 Using Disk as Cache for Tape Backups

The fast 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.

12.4.1.1 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.

Data file image copies can be rolled forward with the latest incremental backups, thereby providing up-to-date data file 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 data file 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 fast 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 is 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 Oracle Data Guard setup that are known to recovery catalog. For `RESYNC CATALOG FROM DB_UNIQUE_NAME` to work, RMAN must 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 is no roll forward because there is no incremental level 1 yet. A level 0 incremental is 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` is created by the `BACKUP DEVICE TYPE DISK ... DATABASE` command. On the third and following days, the roll forward is 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 is a level 0 incremental. On the second and following days, this is 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 fast 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.

12.4.1.2 Commands for Weekly Tape Backups Using Disk as Cache

To back up all recovery-related files to tape, use the RMAN `BACKUP RECOVERY FILES` command once a week.

This ensures that all current incremental, image copy, and archived log backups on disk are backed up to tape.

12.4.2 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

12.4.2.1 Commands for Daily Backups Directly to Tape

The RMAN commands used to perform daily backups directly to tape resynchronize the information from all other databases in the Oracle Data Guard environment.

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 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 fast recovery area.

12.4.2.2 Commands for Weekly Backups Directly to Tape

One day a week, perform a weekly backup directly to tape.

Take the following steps:

1. Connect RMAN to the standby database (as target database) and to the recovery catalog.
2. Execute the following RMAN commands:

```
RESYNC CATALOG FROM DB_UNIQUE_NAME ALL;
BACKUP AS BACKUPSET INCREMENTAL LEVEL 0 DATABASE PLUS ARCHIVELOG;
DELETE ARCHIVELOG ALL;
```

These commands resynchronize the information from all other databases in the Oracle 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 fast recovery area.

12.5 Registering and Unregistering Databases in an Oracle Data Guard Environment

Only the primary database must be explicitly registered using the `REGISTER DATABASE` command. 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 an Oracle Data Guard environment, the database information in the recovery catalog can also be removed after you connect to another database in the same Oracle 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 unregistered is also unregistered from the recovery catalog.

12.6 Reporting in an Oracle 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 database in the Oracle 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;
```

12.7 Performing Backup Maintenance in an Oracle Data Guard Environment

The files in an Oracle Data Guard environment (data files, 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 an Oracle 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. The incremental backups on disk associated with that database and any incremental backups on tape are used to roll forward the image copies. Similarly, during recovery operations, only disk backups associated with the database and files on tape are considered as sources for backups.

> **See Also:**
>
> *Oracle Database Backup and Recovery Reference* for detailed information about RMAN commands

12.7.1 Changing Metadata in the Recovery Catalog

The RMAN `CHANGE` command can be used with various operands to change metadata in the recovery catalog.

For example:

- 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 an Oracle 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 the DB_UNIQUE_NAME Initialization Parameter for a Database

  If the value of the `DB_UNIQUE_NAME` initialization parameter changes for a database, then the same change must be made in the Oracle Data Guard environment. The RMAN recovery catalog, after connecting to that database instance, knows 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 the value of the `DB_UNIQUE_NAME` initialization parameter changes for a database, the same change must be made in RMAN so that it is aware of the new `DB_UNIQUE_NAME`. For example, perform the following steps to change the database with `DB_UNIQUE_NAME` of `BOSTON_A` to `BOSTON_B`:

  1. In the initialization parameter file or SQL, change the `DB_UNIQUE_NAME` initialization parameter from `BOSTON_A` to `BOSTON_B`.
  2. In RMAN, connect to any database in the Oracle Data Guard environment as target database and connect to the recovery catalog. Then execute the `CHANGE` command:

     ```
     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

12.7.2 Deleting Archived Logs or Backups

Use the RMAN `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). 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

12.7.3 Validating Recovery Catalog Metadata

Use the `CROSSCHECK` command to validate and update file status in the recovery catalog schema.

Metadata for all files associated with the current target database (or for any files that are not associated with any database), is 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

12.8 Recovery Scenarios in an Oracle Data Guard Environment

These are some of recovery scenarios that can occur in an Oracle Data Guard environment.

- Recovery from Loss of Files on the Primary or Standby Database
- Recovery from Loss of Online Redo Log Files
- Incomplete Recovery of the Primary Database
- Actions Needed on Standby After TSPITR or Tablespace Plugin at Primary

12.8.1 Recovery from Loss of Files on the Primary or Standby Database

You can restore and recover files over the network by connecting to a physical standby database that contains the required files.

This can be useful when you want to restore lost data files, control files, or tablespaces on a primary database using the corresponding files on the physical standby database. You can also use the same process to restore files on a physical standby database by using the primary database.

For an example of how to restore and recover files by connecting over the network, see *Oracle Database Backup and Recovery User's Guide*.

> **Note:**
>
> In releases prior to Oracle Database 12*c*, to recover from loss of files on the primary, you used the RMAN recovery catalog, and the RMAN `BACKUP`, `CATALOG DATAFILE`, and `SWITCH DATAFILE` commands. To recover from loss of files on the standby, you used the `RESTORE` and `RECOVER` commands. Those methods are no longer necessary as of Oracle Database 12*c*. If you need information about using them, refer to Oracle Database 11*g* documentation.

> **See Also:**
>
> - *Oracle Database Backup and Recovery User's Guide* for more information about using RMAN to restore and recover files over the network

12.8.2 Recovery from Loss of Online Redo Log Files

If all online log members for the current `ACTIVE` group or for an inactive group which has not yet been archived are lost, then you must fail over to the standby database.

Refer to Role Transitions for the failover procedure.

For information about how to recover from the loss of online redo log files in other circumstances, see *Oracle Database Backup and Recovery User's Guide*.

12.8.3 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 data file 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 Using Flashback Database After Issuing an Open Resetlogs Statement 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. Be sure to stop the managed redo process (MRP) before issuing the following command:

   ```
   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. If you recover the database too far (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 Recovering Through the OPEN RESETLOGS Statement 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 Oracle Data Guard configuration.

12.8.4 Actions Needed on Standby After TSPITR or Tablespace Plugin at Primary

After an RMAN tablespace point-in-time recovery (TSPITR) is performed at the primary, the recovered data files have a new system change number (SCN), and are therefore treated like new data files at the primary.

These data files cannot be automatically created at the standby.

Likewise, when a new plugged in tablespace is added to the primary database, the data files are treated like new data files at the primary.

The managed redo process (MRP) at the standby stops when the Redo Apply process encounters creation of these new files. The required new data files must be copied and restored to the standby. You can do this using either backups or a direct copy from the primary. For example, to copy all files that belong to a tablespace that has undergone an RMAN TSPITR, you can use the following RMAN command:

```
RMAN> RESTORE TABLESPACE <tbs_name1, tbs_name2> FROM SERVICE <tnsalias-of-primary>
```

The number of disk channels allocated is per RMAN configurations. So, if `CONFIGURE DEVICE TYPE DISK PARALLELISM 4` is executed, then 4 disk channels are used to pull the files from the primary database.

When the new data files are available at the standby, restart the MRP to continue applying the logs.

> **See Also:**
>
> - *Oracle Database Backup and Recovery User's Guide* for more information about RMAN TSPITR

12.9 Additional Backup Situations

You can 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.

12.9.1 Standby Databases Too Geographically Distant to Share Backups

If the standby databases are far apart from one another, then 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 fast recovery area can reside locally on the primary and standby systems; it 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 Recovery Scenarios in an Oracle Data Guard Environment, 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 fetches the remaining archived redo log files.

12.9.2 Standby Database Does Not Contain Data Files, Used as a 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.

Use the same procedure described in Backup Procedures, with the exception that the RMAN commands that back up database files cannot be run against the FAL server.

12.9.3 Standby Database File Names Are Different From Primary Database

As of Oracle Database 11*g*, the recovery catalog can resynchronize the file names from each standby database site.

However, if the database filenames are not the same on the primary and standby databases that were never resynchronized, then the `RESTORE` and `RECOVER` commands you use are slightly different. To obtain the actual data file names on the standby database, query the `V$DATAFILE` view and specify the `SET NEWNAME` option for all the data files 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, you use the `SET NEWNAME` option of the RMAN `DUPLICATE` command to specify new filenames during standby database creation. Or you could set the `LOG_FILE_NAME_CONVERT` and `DB_FILE_NAME_CONVERT` parameters.

> **See Also:**
>
> Creating a Standby Database That Uses OMF or Oracle ASM for information about precedence rules when both the `DB_FILE_NAME_CONVERT` and `DB_CREATE_FILE_DEST` parameters are set on the standby

12.10 Restoring and Recovering Files Over the Network

As of Oracle Database 12*c*, RMAN lets you restore or recover files by connecting, over the network, to a physical standby database that contains the required files.

You can restore an entire database, data files, control files, spfile, or tablespaces. Restoring files over the network is very useful in scenarios where you need to synchronize the primary and standby databases.

RMAN restores database files, over the network, from a physical standby database by using the `FROM SERVICE` clause of the `RESTORE` command. The `FROM SERVICE` clause provides the service name of the physical standby database from which the files must be restored. During the restore operation, RMAN creates backup sets, on the physical standby database, of the files that need to be restored and then transfers these backup sets to the target database over the network.

> **Note:**
>
> In releases prior to Oracle Database 12*c*, to restore and recover files over the network, you used the RMAN `BACKUP INCREMENTAL FROM SCN` command to create a backup on the primary database that started at the current SCN of the standby, and was then used to roll the standby database forward in time. That manual, multi-step method is not necessary as of Oracle Database 12*c*. If you need information about using that method, refer to Oracle Database 11*g* documentation.

> **See Also:**
>
> - *Oracle Database Backup and Recovery User's Guide* for more information about using RMAN to restore and recover files over the network
> - My Oracle Support note 2005729.1 at `http://support.oracle.com` for information about reducing transportable tablespace downtime using cross-platform incremental backups.

12.11 RMAN Support for CDBs In an Oracle Data Guard Environment

RMAN supports point-in-time recovery (PITR) of a multitenant container database (CDB) at a standby. (Individual pluggable databases (PDBs) do not have their own individual standbys.)

This is in addition to the support RMAN provides for complete database recovery and complete data file recovery at a standby.

To perform a CDB PITR at a standby, connect to the CDB as root and issue the RMAN `BACKUP`, `RESTORE`, and `RECOVER` commands as necessary.

Be aware that when a CDB PITR is performed on a standby, any pluggable databases (PDBs) that were in a disabled state before the CDB PITR become enabled. To return a PDB to a disabled state, connect to it, ensure it is closed (the `OPEN_MODE` column in the `V$PDBS` view shows a value of `MOUNTED`), and then execute the SQL statement `ALTER PLUGGABLE DATABASE DISABLE RECOVERY`.

The `ALTER PLUGGABLE DATABASE DISABLE RECOVERY` statement takes all data files belonging to the PDB offline and disables recovery for the PDB. The data files that belong to the PDB are not part of any recovery session until the PDB is enabled again. Any new

data files created while recovery is disabled are created as unnamed files and are marked offline.

To bring all data files that belong to a PDB back online and enable it for recovery, connect to it, ensure it is closed (the `OPEN_MODE` column in the `V$PDBS` view shows a value of `MOUNTED`), and issue the SQL statement `ALTER PLUGGABLE DATABASE ENABLE RECOVERY`.

To check whether recovery is enabled or disabled on a PDB, query the `V$PDBS` view as follows:

```
SQL> SELECT RECOVERY_STATUS FROM V$PDBS;
```

Flashing Back a PDB

As of Oracle Database 12*c* Release 2 (12.2.0.1), you can use the `FLASHBACK PLUGGABLE DATABASE` command (available through SQL or Recovery Manager) to perform a flashback operation on a specific PDB. You can flashback to a specific restore point — an alias for system change number (SCN)— in the past without affecting other PDBs in a CDB. (Performing such an operation on a PDB is analogous to `FLASHBACK DATABASE` in a non-CDB.)

You can also flashback a PDB on a standby. In effect, flashing back a PDB on a standby rewinds the data files for the PDB to a previous point in time, as if restoring a backup of the PDB. Flashing back a PDB on a standby allows the standby to quickly follow the primary after you perform a PDB PITR/flashback operation on the primary, as described in Actions Needed On a Standby After a PDB PITR On a Primary.

Note:

Files that are brought online or offline as a result of an `ALTER PLUGGABLE DATABASE [ENABLE | DISABLE]` operation remain in that state even if you flashback the database to a point before the operation was performed.

See Also:

- Actions Needed On a Standby After a PDB PITR On a Primary for information about actions needed on a standby after a PDB PITR on a primary
- Actions Needed on Standby After TSPITR or Tablespace Plugin at Primary for information about actions needed on a standby after TSPITR or tablespace plugin at the primary

13
Using SQL Apply to Upgrade the Oracle Database

You can use a logical standby database to perform a *rolling upgrade* of Oracle Database software.

During a rolling upgrade, you can run different releases of Oracle Database on the primary and logical standby databases while you upgrade them, one at a time, incurring minimal downtime on the primary database.

For databases originating with the first patch set of Oracle Database 12*c* Release 1 (12.1), the preferred method for performing a rolling upgrade with an existing physical standby database is to use the `DBMS_ROLLING` PL/SQL package, as described in Using DBMS_ROLLING to Perform a Rolling Upgrade .

The following topics describe how to minimize downtime while upgrading an Oracle database:

- 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

> **Note:**
>
> These topics describe an alternative to the usual upgrade procedure involving longer downtime, as described in Upgrading and Downgrading Databases in an Oracle Data Guard Configuration. Do not attempt to combine steps from the two procedures.

13.1 Benefits of a Rolling Upgrade Using SQL Apply

These are the advantages of performing a rolling upgrade with SQL Apply.

- Your production database incurs 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.
- A logical standby accepts archived logs while the upgrade is taking place, which provides an added level of disaster protection.

Note:

- As of Oracle Database 12*c* Release 1 (12.1), Oracle XML DB Repository supports Oracle Data Guard rolling upgrades. See *Oracle XML DB Developer's Guide* for more information about considerations and restrictions to keep in mind with regard to this support.
- As of Oracle Database 12*c* Release 1 (12.1), you can upgrade databases that use Oracle Database Vault to new Oracle Database releases and patch sets by using Oracle Data Guard database rolling upgrades with a transient logical standby and the PL/SQL package, `DBMS_ROLLING`.
- As of Oracle Database 12*c* Release 1 (12.1), you can upgrade databases that use Oracle Label Security (OLS) to new Oracle Database releases and patch sets by using Oracle Data Guard database rolling upgrades using a transient logical standby database and the PL/SQL package, `DBMS_ROLLING`.

13.2 Requirements to Perform a Rolling Upgrade Using SQL Apply

These are the requirements for performing a rolling upgrade using SQL Apply.

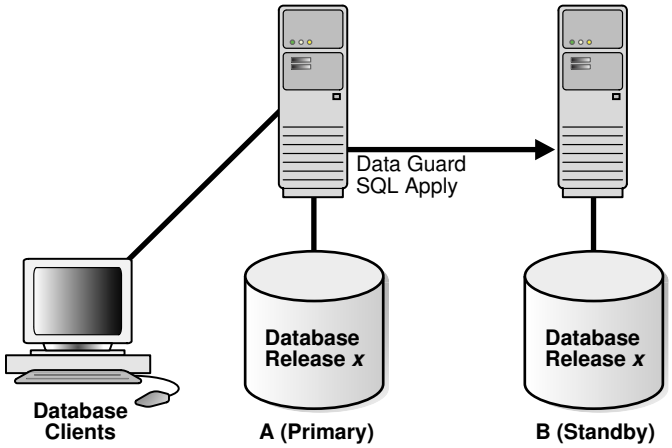
- If the database is part of an Oracle Data Guard broker configuration, then disable the broker configuration before the rolling upgrade. See *Oracle Data Guard Broker* for information about disabling a broker configuration.
- The Oracle Data Guard protection mode must be set to either maximum availability 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 set on the primary database must match the software release prior to the upgrade. Therefore, a rolling upgrade from release *x* to release *y* requires that the `COMPATIBLE` initialization parameter be set to release *x* on the primary database. The rolling upgrade standby database must have its `COMPATIBLE` initialization parameter set to *x* or higher.

13.3 Figures and Conventions Used in the Upgrade Instructions

This background information will help you better understand instructions given for performing upgrades.

Figure 13-1 shows an Oracle Data Guard configuration before the upgrade begins, with the primary and logical standby databases both running the same Oracle Database software release.

Figure 13-1 Oracle Data Guard Configuration Before Upgrade



During the upgrade process, the Oracle 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 Oracle 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 13-1.

The following sections describe scenarios in which you can use the SQL Apply rolling upgrade procedure:

- 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

13.4 Performing a Rolling Upgrade By Creating a New Logical Standby Database

This scenario assumes you do not have an existing Oracle 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.

Table 13-1 lists the steps to prepare the primary and standby databases for upgrading.

Table 13-1 Steps to Perform a Rolling Upgrade by Creating a New Logical Standby

Step	Description
Step 1	Identify unsupported data types and storage attributes
Step 2	Create a logical standby database

Table 13-1 (Cont.) Steps to Perform a Rolling Upgrade by Creating a New Logical Standby

Step	Description
Step 3	Perform a rolling upgrade

1. Identify unsupported database objects on the primary database and decide how to handle them by doing the following:
   - Review the list of supported data types and storage attributes provided in 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 are not applied on the logical standby database. Use the following query to see a list of unsupported tables:

     ```
     SQL> SELECT DISTINCT OWNER, TABLE_NAME FROM DBA_LOGSTDBY_UNSUPPORTED;
     ```

     Use the following query to see a list of unsupported internal schemas:

     ```
     SQL> SELECT OWNER FROM DBA_LOGSTDBY_SKIP -
     >  WHERE STATEMENT_OPT = 'INTERNAL SCHEMA';
     ```
2. Create a logical standby database, following the instructions in Creating a Logical Standby Database .

> **Note:**
>
> Before you start SQL Apply for the first time, make sure you 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` view:
>
> ```
> EXECUTE DBMS_LOGSTDBY.APPLY_SET('MAX_EVENTS_RECORDED',
>  DBMS_LOGSTDBY.MAX_EVENTS);
>
> EXECUTE DBMS_LOGSTDBY.APPLY_SET('RECORD_UNSUPPORTED_OPERATIONS',
>  'TRUE');
> ```

   Oracle recommends configuring a standby redo log on the logical standby database to minimize downtime.
3. Perform a rolling upgrade now that you have created a logical standby database. Follow the procedure described in Performing a Rolling Upgrade With an Existing Logical Standby Database, which assumes that you have a logical standby running the same Oracle software.

13.5 Performing a Rolling Upgrade With an Existing Logical Standby Database

These steps show how to perform a rolling upgrade of Oracle Database on the logical standby database and the primary database.

Table 13-2 lists the steps.

Table 13-2 Steps to Perform a Rolling Upgrade With an Existing Logical Standby

Step	Description
Step 1	Prepare for rolling upgrade
Step 2	Upgrade the logical standby database
Step 3	Restart SQL Apply on the upgraded logical standby database
Step 4	Monitor events on the upgraded standby database
Step 5	Begin a switchover
Step 6	Import any tables that were modified during the upgrade
Step 7	Complete the switchover and activate user applications
Step 8	Upgrade the old primary database
Step 9	Start SQL Apply on the old primary database
Step 10	Optionally, raise the compatibility level on both databases
Step 11	Monitor events on the new logical standby database
Step 12	Optionally, perform another switchover

1. Follow these steps to prepare to perform a rolling upgrade of Oracle Software:
   a. Stop SQL Apply by issuing the following statement on the logical standby database (Database B):

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

   b. Set compatibility, if needed, to the highest value.

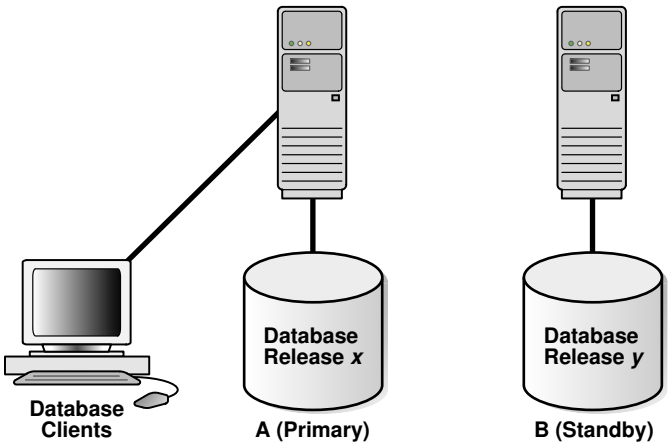
      Ensure the `COMPATIBLE` initialization parameter specifies the release number for the Oracle Database software running on the primary database prior to the upgrade.

      For example, if the primary database is running release 11.1, then set the `COMPATIBLE` initialization parameter to 11.1 on both databases. Be sure to set the `COMPATIBLE` initialization parameter on the standby database first *before* you set it on the primary database.
2. Upgrade Oracle database software on the logical standby database (Database B) to release *y*. While the logical standby database is being upgraded, it does not accept redo data from the primary database.

   To upgrade Oracle Database software, refer to the *Oracle Database Upgrade Guide* for the applicable Oracle Database release.

   Figure 13-2 shows Database A running release *x*, and Database B running release *y*. During the upgrade, redo data accumulates on the primary system.

Figure 13-2 Upgrade the Logical Standby Database Release

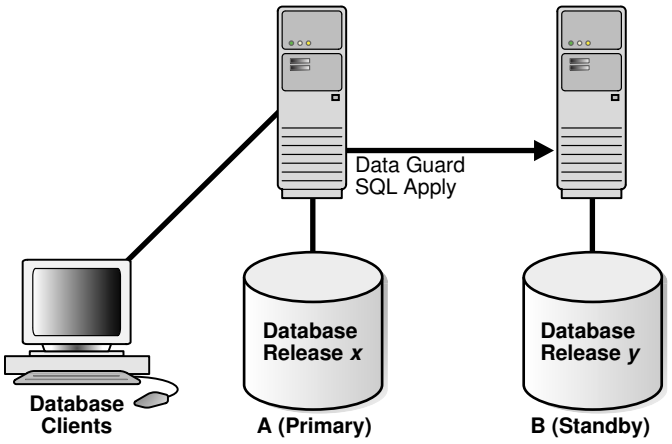


3. Restart SQL Apply and operate with release *x* on Database A and release *y* on Database B. To start SQL Apply, issue the following statement on Database B:

```
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 Oracle Data Guard configuration can run the mixed releases shown in Figure 13-3 for an arbitrary period while you verify that the upgraded Oracle Database software release is running properly in the production environment.

Figure 13-3 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
```

4. It is recommended that you 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.

```
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.

5. 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 Performing a Switchover to a Logical Standby Database 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 a 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.

6. Step 4 described how to list unsupported tables that are being modified. If unsupported DML statements were issued on the primary database, then 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_EXISTS_ACTION=TRUNCATE
   ```

   This command prompts you for the `impdp` password before executing.

7. When you are satisfied that the upgraded database software is operating properly, complete the switchover to reverse the database roles:

   a. 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
      ```

   b. 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 PRIMARY;
      ```

> **Note:**
>
> The usual two-phased prepared switchover described in Performing a Switchover to a Logical Standby Database 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 a logical standby database.

c. Activate the user applications and services on Database B, which is now running in the primary database role.

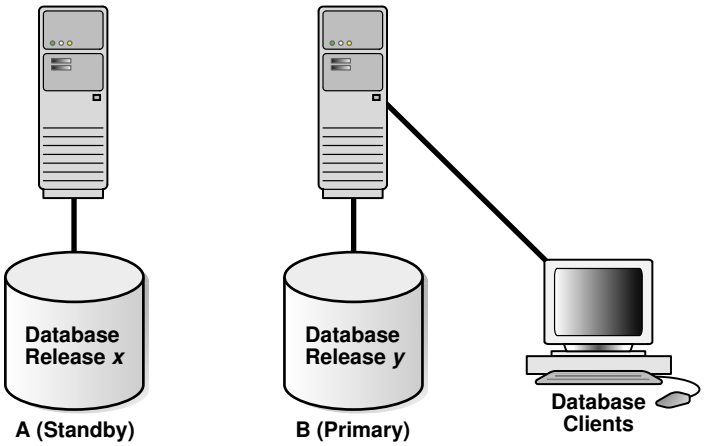
After the switchover, you cannot send redo data from the new primary database (B) that is running the new database software release to the new standby database (A) that is running an older software release. This means the following:

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

Figure 13-4 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 8.

Figure 13-4 After a Switchover

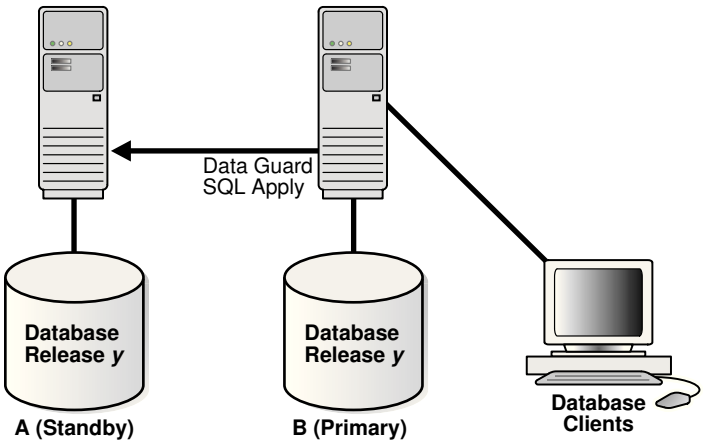


8. Database A is still running release *x* and cannot apply redo data from Database B until you upgrade it and start SQL Apply.

For more information about upgrading Oracle Database software, see the *Oracle Database Upgrade Guide*.

Figure 13-5 shows the system after both databases have been upgraded.

Figure 13-5 Both Databases Upgraded



9. 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 must 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 must connect as user `SYSTEM` or with an account with a similar level of privileges.

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.

10. 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.

11. 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 4.

    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 Creating a Logical Standby Database for complete information.

12. Optionally, perform another switchover of the databases so Database A is once again running in the primary database role (as shown in Figure 13-1).

> **Note:**
>
> You use the two-phased prepared switchover described in Performing a Switchover to a Logical Standby Database since at this time, both Database A and Database B are running the same version of the Oracle software.

13.6 Performing a Rolling Upgrade With an Existing Physical Standby Database

These steps show how to perform a rolling upgrade of Oracle Database 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:**
>
> These steps 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 13-3 summarizes the steps involved.

Table 13-3 Steps to Perform a Rolling Upgrade With an Existing Physical Standby

Step	Description
Step 1	Prepare the primary database for a rolling upgrade (perform these steps on Database A)
Step 2	Convert the physical standby database into a logical standby database (perform these steps on Database B)
Step 3	Upgrade the logical standby database and catch up with the primary database (perform these steps on Database B)
Step 4	Flashback Database A to the guaranteed restore point (perform these steps on Database A)
Step 5	Mount Database A using the new version of Oracle software
Step 6	Convert Database A to a physical standby
Step 7	Start managed recovery on Database A
Step 8	Perform a switchover to make Database A the primary database
Step 9	Clean up the guaranteed restore point created in Database A

1. Prepare the primary database for a rolling upgrade (perform these steps on Database A)
   a. Enable Flashback Database, if it is not already enabled:

```
SQL> SHUTDOWN IMMEDIATE;
SQL> STARTUP MOUNT;
SQL> ALTER DATABASE FLASHBACK ON;
SQL> ALTER DATABASE OPEN;
```

b. Create a guaranteed restore point:

```
SQL> CREATE RESTORE POINT pre_upgrade GUARANTEE FLASHBACK DATABASE;
```

c. (Optional) Identify unsupported data types by performing the following SQL query on the primary database:

```
SELECT * FROM DBA_LOGSTDBY_EDS_SUPPORTED;
```

For any tables returned that have unsupported data types, you can use the Extended Datatype Support (EDS) feature to replicate them. To do so, execute the following PL/SQL procedure for each table to be replicated using EDS:

```
SQL> EXECUTE DBMS_LOGSTDBY.EDS_ADD_TABLE(schema_name, table_name);
```

Because these steps are done on the primary, there is no need to invoke the procedure on the physical standby because the EDS objects created by `EDS_ADD_TABLE` are replicated to the physical standby. Once the physical standby is converted to a logical standby, EDS replication is enabled automatically and the tables are maintained by SQL Apply.

2. Convert the physical standby database into a logical standby database (perform these steps on Database B).

a. Follow the steps outlined in Creating a Logical Standby Database except for the following difference. In 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;
```

b. You must take the following actions before you start SQL Apply for the first time:

i. Disable automatic deletion of foreign archived logs at the logical standby, as follows:

```
SQL> EXECUTE DBMS_LOGSTDBY.APPLY_SET('LOG_AUTO_DELETE', 'FALSE');
```

> **Note:**
>
> Do 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.

ii. Make sure you 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:

```
SQL> EXECUTE DBMS_LOGSTDBY.APPLY_SET('MAX_EVENTS_RECORDED', -
> DBMS_LOGSTDBY.MAX_EVENTS);

SQL> EXECUTE DBMS_LOGSTDBY.APPLY_SET('RECORD_UNSUPPORTED_OPERATIONS',
'TRUE');
```

iii. Start SQL Apply for the first time, as follows:

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

> **See Also:**
>
> - Customizing Logging of Events in the DBA_LOGSTDBY_EVENTS View 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

3. You can now follow Steps 1 through 7 as described in Performing a Rolling Upgrade With an Existing Logical Standby Database . At the end of these steps, Database B is your primary database running the upgraded version of the Oracle software, and Database A becomes your logical standby database.

   If you executed DBMS_LOGSTDBY.EDS_ADD_TABLE in step 1, you can now execute DBMS_LOGSTDBY.EDS_REMOVE_TABLE at the current primary database (Database B).

   Move on to the next step to turn Database A into the physical standby for Database B.

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;
   ```

5. At this point, switch Database A to use the higher version of the Oracle software. You do not run the upgrade scripts, since Database A is turned into a physical standby, and upgraded automatically as it applies the redo data generated by Database B.

   Mount Database A, as follows:

   ```
   SQL> STARTUP MOUNT;
   ```

6. Convert Database A to a physical standby.

   ```
   SQL> ALTER DATABASE CONVERT TO PHYSICAL STANDBY;
   SQL> SHUTDOWN IMMEDIATE;
   ```

7. Start managed recovery on Database A.

   Database A is upgraded automatically as it applies the redo data generated by Database B. Managed recovery waits 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 -
>  DISCONNECT FROM SESSION;
```

> **Note:**
>
> When Redo Apply restarts, it waits for a new incarnation from the current primary database (Database B) to be registered.

8. 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 "Performing a Switchover to a Physical Standby Database".
9. To preserve disk space, drop the existing guaranteed restore point created in Database A:

```
SQL> DROP RESTORE POINT PRE_UPGRADE;
```

> **See Also:**
>
> The "Database Rolling Upgrade Using Transient Logical Standby: Oracle Data Guard 11*g*" best practices white paper available on the Oracle Maximum Availability Architecture (MAA) home page at:
>
> `http://www.oracle.com/goto/maa`

14

Using DBMS_ROLLING to Perform a Rolling Upgrade

The Rolling Upgrade Using Oracle Active Data Guard feature provides a streamlined method of performing rolling upgrades.

It is implemented using the `DBMS_ROLLING` PL/SQL package, which enables you to upgrade the database software in an Oracle Data Guard configuration in a rolling fashion. The Rolling Upgrade Using Oracle Active Data Guard feature requires a license for the Oracle Active Data Guard option.

You can use this feature to perform database version upgrades starting with the first patchset of Oracle Database 12*c*. You cannot use it to upgrade from any version earlier than the first Oracle Database 12*c* patchset. This means that the manual Transient Logical Standby upgrade procedure must still be used when upgrading from Oracle Database 11*g* to Oracle Database12*c,* or when upgrading from the initial Oracle Database 12*c* release to the first patchset of Oracle Database 12*c*.

The `DBMS_ROLLING` package performs Oracle Data Guard switchovers to minimize downtime of the primary database service. Prior to using `DBMS_ROLLING`, the Oracle Data Guard environment must be properly configured to accommodate switchovers. The setup requirements differ depending on whether Oracle Data Guard broker is active during execution of `DBMS_ROLLING`:

- If the broker is going to be active during `DBMS_ROLLING`, then see *Oracle Data Guard Broker* for information about setting up the broker for a switchover.
- If the broker is not going to be active during `DBMS_ROLLING`, then see Role Transitions Involving Logical Standby Databases. The absence of Oracle Data Guard broker means that `LOG_ARCHIVE_DEST_`*n* parameters must be properly configured on the target primary database so that redo shipping resumes after the switchover.

Additionally, you can use this feature immediately for other database maintenance tasks beginning with Oracle Database 12*c* Release 1 (12.1). The database where maintenance is performed must be operating at a minimum of Oracle 12.1. Such maintenance tasks include:

- Adding partitioning to non-partitioned tables
- Changing BasicFiles LOBs to SecureFiles LOBs
- Changing `XMLType` stored as `CLOB` to `XMLtype` stored as binary XML
- Altering tables to be OLTP-compressed

See the following topics:

- Concepts New to Rolling Upgrades
- DBMS_ROLLING Upgrades and CDBs
- Overview of Using DBMS_ROLLING
- Planning a Rolling Upgrade

14.1 Concepts New to Rolling Upgrades

To upgrade the database software in an Oracle Data Guard configuration in a rolling fashion, you first designate a physical standby as the future primary database.

Conceptually, the rolling upgrade process splits the Oracle Data Guard configuration into two groups: the leading group (LG) and the trailing group (TG).

Databases in the leading group are upgraded first; hence the name leading group. The leading group contains the designated future primary database, and the physical standbys that you can configure to protect the designated future primary. The future primary is first converted into a logical standby database and then the new database software is installed on it and the upgrade process is run. Other standby databases in the leading group also must have their software upgraded at this point.

The trailing group contains the original primary database and standby databases that protect the original primary during the rolling upgrade process. While the databases in the leading group are going through the upgrade process, user applications can still be connected to the original primary and making changes. The trailing group databases continue running the old database software until all the databases in the leading group are upgraded and the future primary has caught up with the original primary by applying the changes that were generated at the original primary database during the upgrade window. At this point a switchover is done to transfer the primary role to the designated future primary database, and the user applications are switched over to the new primary database. New software is then installed on the databases that are part of the trailing group, and they are reinstated into the configuration as standbys to the new primary database.

The standbys in the respective groups are called the Leading Group Standbys (LGS) and Trailing Group Standbys (TGS). Other than the designated future primary, all other standbys in the leading group can only be physical standbys. The trailing group can contain both physical and logical standbys; they are called Trailing Group Physical (TGP) and Trailing Group Logical (TGL) in cases where it is necessary to make a distinction between the standby types. The designated future primary is also called the Leading Group Master (LGM) and the original primary database is called the Trailing Group Master (TGM).

The `DBMS_ROLLING` package increases the robustness of the rolling upgrade process as follows:

- It can handle failures during the rolling upgrade process. The original primary or the TGM database can fail. You can initiate a regular failover operation to any other physical standby in the trailing group, and then designate the new primary database as the TGM.
- It allows data protection of the LGM (the designated future primary) during the rolling upgrade process. You can set up physical standbys for the LGM database, and thus protect it during the upgrade process and also achieve Zero Data Loss after the upgrade. After the LGM has been successfully upgraded, a failure in the

LGM can be accommodated by failing over to any of its physical standby databases. You can then designate the failover target database to take over the role of the LGM.

Table 14-1 compares the characteristics of TGP standbys versus LGP standbys before and after a switchover operation.

Table 14-1 Trailing Group Physicals (TGP) Versus Leading Group Physicals (LGP)

Standby Type	Before Switchover	After Switchover	Notes
Trailing Group Physical (TGP)	Low apply lag Lower data loss risk	High apply lag Higher data loss risk	Can fail over to the primary role Must flash back like the original primary
Leading Group Physical (LGP)	High apply lag Higher data loss risk	Low apply lag Lower data loss risk	Can fail over to the transient logical standby role Does not have to flash back like the original primary

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for a description of the `DBMS_ROLLING` PL/SQL package
- Unsupported Tables During Rolling Upgrades for information about how to determine whether any of the tables involved in the upgrade contain data types that are unsupported when performing an upgrade using the `DBMS_ROLLING` PL/SQL package
- Additional PL/SQL Package Support Available Only in the Context of DBMS_ROLLING Upgrades for information about PL/SQL packages that are supported only in the context of a `DBMS_ROLLING` upgrade

14.1.1 Data Guard Broker Support for DBMS_ROLLING Upgrades

As of Oracle Database 12*c* Release 2 (12.2.0.1), Data Guard broker can remain on during a `DBMS_ROLLING` rolling upgrade; there is no longer any need to disable it.

However, the fast-start failover feature must be disabled before starting a `DBMS_ROLLING` upgrade and any attempts to enable fast-start failover while the rolling upgrade is in progress are rejected. In addition, while a rolling upgrade is in progress, role changes are permissible only to the standby databases that are protecting the current primary database. The broker reports the role of the rolling upgrade target as `Transient Logical Standby` during a `SHOW CONFIGURATION` command as well as reporting the configuration status as `ROLLING DATABASE MAINTENANCE IS IN PROGRESS`. If there are standby databases protecting both the original primary and the upgrade target, then this topology is reflected when the `SHOW CONFIGURATION` command is issued from the current primary as well as from the upgrade target (before it has taken over as the primary database).

Broker support is enabled by default during execution of the `DBMS_ROLLING.BUILD_PLAN` procedure if the broker is enabled at the time of the call. When broker support is enabled, the broker sets up the redo transport destinations as necessary from the

original primary database as well as from the rolling upgrade target, manages instance count on the upgrade target if it is an Oracle RAC database, and notifies Oracle Clusterware and Global Data Services as appropriate during the course of the rolling upgrade. Broker support can be manually controlled using the `DBMS_ROLLING` parameter, `DGBROKER`.

Although role transitions are typically performed using the broker, the switchover step in a rolling upgrade should continue to be performed using the `DBMS_ROLLING.SWITCHOVER` procedure.

Information about the status of a rolling upgrade being done using the PL/SQL package `DBMS_ROLLING`, is displayed in the output of the broker commands `SHOW CONFIGURATION` and `SHOW DATABASE`.

The `SHOW CONFIGURATION` command shows Transient logical standby database as the role of the upgrade target, and `ROLLING DATABASE MAINTENANCE IN PROGRESS` as the configuration status. An example of this output is as follows:

```
Configuration - DRSolution
Protection Mode: MaxPerformance
Members:
North_Sales - Primary database
South_Sales - Transient logical standby database
Fast-Start Failover: DISABLED
Configuration Status:
ROLLING DATABASE MAINTENANCE IN PROGRESS
```

The `SHOW DATABASE` command shows a `WARNING` with an appropriate `ORA` error for the upgrade target and the trailing or leading standbys, depending on the current rolling upgrade progress. An example of this output is as follows:

```
Database - South_Sales
Role: Physical standby database
Intended State: APPLY-ON
Transport Lag: ***
Apply Lag: ***
Average Apply Rate: ***
Real Time Query: OFF
Instance(s):
South
Database Warning(s):
ORA-16866: database converted to transient logical standby database for rolling
database maintenance
Database Status:
WARNING
```

See Also:

- DBMS_ROLLING
- SHOW CONFIGURATION
- SHOW DATABASE

14.2 DBMS_ROLLING Upgrades and CDBs

The steps to performing a rolling upgrade using DBMS_ROLLING on a multitenant container database (CDB) are no different from non-CDB environments, but there are additional requirements and considerations.

Requirements Specific to DBMS_ROLLING Upgrades on a CDB

The additional requirements when you use DBMS_ROLLING to perform a rolling upgrade on a CDB are as follows.

- The TNS services referenced in the LOG_ARCHIVE_DEST_n parameters must be services that resolve to the root container of the destination database. The process assisting DBMS_ROLLING performs numerous operations which can only execute from the root container.
- All container databases on the transient logical standby must be plugged in and opened prior to calling DBMS_ROLLING.SWITCHOVER. This eliminates the possibility that the logical standby apply engine will halt because it cannot apply to a given PDB.

See Example 14-5 for an example rolling upgrade using DBMS_ROLLING.

Additional Considerations for DBMS_ROLLING Upgrades on a CDB

Installing, upgrading, or patching of applications installed in application containers is not supported while a DBMS_ROLLING upgrade is in progress.

- If DDL is executed to start the install, upgrade, or patching of an application container while a DBMS_ROLLING upgrade is in progress, then an error is returned. (The DBMS_ROLLING upgrade continues.)
- If an upgrade to an application container is in progress, then an attempt to start a DBMS_ROLLING upgrade results in an error. (The application container upgrade continues.)
- If a DBMS_ROLLING upgrade is performed and database compatibility is set to 12.2 or higher, then replication of application containers is supported. Except for DBMS_ROLLING upgrades, logical standby does not offer any support for application containers; such containers are skipped and a message is written to the alert log indicating that application containers are being skipped.
- The CDB that you are upgrading using DBMS_ROLLING can contain pluggable databases (PDBs) with different character sets.

14.3 Overview of Using DBMS_ROLLING

There are three stages to the rolling upgrade process using the DBMS_ROLLING PL/SQL Package: specification, compilation, and execution.

1. **Specification**: You first specify how you want to implement the rolling upgrade process. It is mandatory that you designate a future primary database. This act conceptually creates the leading and the trailing groups. At this point, the leading group only contains the LGM. You can optionally specify other standbys to protect the LGM.

   You use the following procedures during the specification phase:

- `DBMS_ROLLING.INIT_PLAN`
- `DBMS_ROLLING.SET_PARAMETER`

2. **Compilation**: This is initiated by calling the `DBMS_ROLLING.BUILD_PLAN` procedure. The `BUILD_PLAN` procedure communicates with all databases participating in the rolling upgrade and assembles a rolling upgrade plan. The `BUILD_PLAN` procedure is also called to alter an existing rolling upgrade plan. Alterations are necessary after changes to `DBMS_ROLLING` parameters and after changes to the topology as a result of failover. All participating databases must be reachable during execution of the `BUILD_PLAN` procedure because numerous validations are required to ensure a successful rolling upgrade.
3. **Execution**: Execution of the rolling upgrade has five stages.

   Stage 1: The `DBMS_ROLLING.START_PLAN` procedure starts the execution of the rolling upgrade. This converts the LGM database to a logical standby and starts the SQL Apply process at the LGM.

   Stage 2: You upgrade the database software at the databases that are part of the leading group. You also run the upgrade scripts at the LGM. After this is done, you must restart SQL Apply processes at the LGM database. (See *Oracle Database Upgrade Guide* for information about upgrade scripts.) Leading group physical standbys are also addressed during this stage by re-mounting them using the higher version binaries. These databases are upgraded via recovery of the redo from the LGM.

   Stage 3: After the apply lag reaches a given threshold (set to 10 minutes by default, but can be configured during the specification stage), the `DBMS_ROLLING.SWITCHOVER` procedure proceeds with the switchover operation. When the switchover is complete, the LGM becomes the primary database.

   Stage 4: The LGM is now the primary database running the new database software and the databases in the leading group are protecting it. The TGM is mounted and the databases in the trailing group are still running the older version of the database software. You must prepare the TGM and TGS databases for upgrade by upgrading the database software and re-mounting the databases on the higher version binaries. (See *Oracle Database Upgrade Guide* for information about upgrade scripts.)

   Stage 5: Execute the `DBMS_ROLLING.FINISH_PLAN` procedure at the current primary database (originally the LGM). It reinstates all the databases in the trailing group to become the standbys of the current primary database, and restarts the apply processes. The `FINISH_PLAN` procedure waits for all databases in the trailing group to be upgraded to the new release (although the database software for the trailing group databases was changed in Stage 4, the data dictionary of the trailing group databases, except for any logical standbys in the trailing group, are updated based on media recovery of the redo generated during the upgrade at the LGM database).

After the rolling upgrade has been successfully executed, you can remove your rolling upgrade specification by calling the `DBMS_ROLLING.DESTROY_PLAN` procedure.

14.4 Planning a Rolling Upgrade

Planning your rolling upgrade is essential to a successful upgrade experience. In the planning phase you specify various upgrade parameters and build an upgrade plan.

The parameters and upgrade plan forecast all the operational details unique to your environment. The upgrade plan performs site-specific validations to alert you to configuration and resource problems which could potentially disrupt the rolling upgrade.

The tasks necessary to define upgrade parameters and build an upgrade plan are as follows:

- Initialize the upgrade parameters
- View the current upgrade parameter values
- Modify the upgrade parameter values, as necessary
- Build the upgrade plan
- View the current upgrade plan
- Revise the upgrade plan, as necessary

The rest of this section describes each of these steps in detail. They must be performed in the order presented.

1. Plan parameters must be initialized to system-generated default values before they can be customized. To initialize plan parameters, call the `DBMS_ROLLING.INIT_PLAN` procedure. This procedure identifies the `DB_UNIQUE_NAME` of the future primary database (the leading group master or LGM). The LGM is converted into a logical standby database as part of the `START_PLAN` procedure call. The following is a sample call to the `INIT_PLAN` procedure in which `boston` is identified as the future primary database:

   ```
   DBMS_ROLLING.INIT_PLAN(future_primary=>'boston');
   ```

   The `INIT_PLAN` procedure returns an initial set of system-generated plan parameters. It adds each physical and logical standby database specified in the `DG_CONFIG` init.ora parameter as a participant in the rolling upgrade. Other databases (such as downstream databases serving GoldenGate downstream deployment or snapshot standbys) are excluded automatically.

   By default, standby databases other than the future primary are configured to protect the primary database, and are configured as mandatory participants in the rolling upgrade.

   Once the database-related parameters have been defined, the `INIT_PLAN` procedure defines operational parameters with system-supplied defaults. In most cases, the plan parameters are ready for plan validation, but to ensure they meet your needs, be sure to review each parameter.

   Plan parameters are persisted in the database until you call the `DESTROY_PLAN` procedure to remove all states related to the rolling upgrade.

2. After the `INIT_PLAN` procedure has completed, you can query the `DBA_ROLLING_PARAMETERS` view to see the plan parameters and their current values. Plan parameters are either global or local in scope. Global parameters are attributes of the rolling upgrade as a whole and are independent of the database

participants. Global parameters have a NULL value in the SCOPE column. Local parameters have a specific database name in the SCOPE column, with which they are associated. The following is a sample query:

```
SQL> select scope, name, curval from dba_rolling_parameters order by scope, name;

SCOPE          NAME                        CURVAL
-------------- ------------------------    ------------------------------
seattle        INVOLVEMENT                 FULL
seattle        MEMBER                      NONE
boston         INVOLVEMENT                 FULL
boston         MEMBER                      TRAILING
oakland        INVOLVEMENT                 FULL
oakland        MEMBER                      TRAILING
atlanta        INVOLVEMENT                 FULL
atlanta        MEMBER                      LEADING
               ACTIVE_SESSIONS_TIMEOUT     3600
               ACTIVE_SESSIONS_WAIT        0
               BACKUP_CONTROLFILE          rolling_change_backup.f
               DGBROKER                    0
               DICTIONARY_LOAD_TIMEOUT     3600
               DICTIONARY_LOAD_WAIT        0
               DICTIONARY_PLS_WAIT_INIT    300
               DICTIONARY_PLS_WAIT_TIMEOUT 3600
               EVENT_RECORDS               10000
               FAILOVER                    0
               GRP_PREFIX                  DBMSRU_
               IGNORE_BUILD_WARNINGS       0
               IGNORE_LAST_ERROR           0
               LAD_ENABLED_TIMEOUT         600
               LOG_LEVEL                   INFO
               READY_LGM_LAG_TIME          600
               READY_LGM_LAG_TIMEOUT       60
               READY_LGM_LAG_WAIT          0
               SWITCH_LGM_LAG_TIME         600
               SWITCH_LGM_LAG_TIMEOUT      60
               SWITCH_LGM_LAG_WAIT         1
               SWITCH_LGS_LAG_TIME         60
               SWITCH_LGS_LAG_TIMEOUT      60
               SWITCH_LGS_LAG_WAIT         0
               UPDATED_LGS_TIMEOUT         10800
               UPDATED_LGS_WAIT            1
               UPDATED_TGS_TIMEOUT         10800
               UPDATED_TGS_WAIT            1
35 rows selected.
```

In the sample output, the databases atlanta, boston, oakland, and seattle were all discovered through the DG_CONFIG, and assigned parameters in the current plan.

> **See Also:**
>
> - *Oracle Database Reference* for more information about the DBA_ROLLING_PARAMETERS view

3. To modify any existing rolling upgrade parameter, use the DBMS_ROLLING.SET_PARAMETER PL/SQL procedure. The following is an example of using the SET_PARAMETER procedure:

```
DBMS_ROLLING.SET_PARAMETER(
  scope IN VARCHAR2,
  name IN VARCHAR2,
  value IN VARCHAR2);
```

The scope identifies either a `DB_UNIQUE_NAME` value for local parameters or `NULL` for global parameters. It is not necessary to provide a scope of `NULL` for parameters that are not specific to a database.

The name is the name of the parameter to modify.

The value identifies the value for the specified parameter. A value of NULL reverts the parameter back to its system-supplied default if one exists.

> **See Also:**
>
> - *Oracle Database PL/SQL Packages and Types Reference* for a complete list of all available rolling upgrade parameters

The following examples illustrate sample usage of some rolling upgrade parameters.

4. After all the necessary parameters are specified, you build an upgrade plan. An upgrade plan is a custom generated set of instructions which guides your Oracle Data Guard configuration through a rolling upgrade. (The plan that is generated differs based on whether or not the configuration is managed by Data Guard broker.)

   To build an upgrade plan, use the `DBA_ROLLING.BUILD_PLAN` PL/SQL procedure. This procedure requires the configuration to be exactly as described by the plan parameters with all of the instances started and reachable through the network.

   The procedure is called as follows:

   ```
   DBMS_ROLLING.BUILD_PLAN;
   ```

   There are no arguments to specify because the procedure gets all its input from the `DBA_ROLLING_PARAMETERS` view. The procedure validates plan parameters and performs site-specific validations of resources such as log transport and flash recovery area settings. In general, configuration settings that do not meet the criteria of best-practice values are treated as warnings and recorded in the `DBA_ROLLING_EVENTS` view. By default, the `IGNORE_BUILD_WARNINGS` parameter is set to `1`, meaning warnings do not prevent an upgrade plan from reaching a usable state. You can set this parameter to `0` for stricter rule enforcement when plans are built.

   > **Note:**
   >
   > The validations performed during plan generation are specific to rolling upgrades. They are not a substitute for the recommended practice of running the Pre-Upgrade Information Tool to evaluate upgrade readiness.

   After generating the plan, move on to the following steps to view it, diagnose any problems with it, and revise it if necessary.

5. After the `BUILD_PLAN` procedure successfully returns, the complete upgrade plan is viewable in the `DBA_ROLLING_PLAN` view. Each record in the view identifies a specific instruction that is scheduled for execution.

   The following output is an example of how a rolling upgrade plan would appear:

```
SQL> SELECT instid, target, phase, description FROM DBA_ROLLING_PLAN;

INSTID TARGET       PHASE   DESCRIPTION
------ ------------ ------- --------------------------------------------------
     1 seattle      START   Verify database is a primary
     2 seattle      START   Verify MAXIMUM PROTECTION is disabled
     3 boston       START   Verify database is a physical standby
     4 boston       START   Verify physical standby is mounted
     5 oakland      START   Verify database is a physical standby
     6 oakland      START   Verify physical standby is mounted
     7 atlanta      START   Verify database is a physical standby
     8 atlanta      START   Verify physical standby is mounted
     9 seattle      START   Verify server parameter file exists and is modifiable
    10 boston       START   Verify server parameter file exists and is modifiable
    11 oakland      START   Verify server parameter file exists and is modifiable
    12 atlanta      START   Verify server parameter file exists and is modifiable
    13 seattle      START   Verify Data Guard Broker configuration is disabled
    14 boston       START   Verify Data Guard Broker configuration is disabled
    15 oakland      START   Verify Data Guard Broker configuration is disabled
    16 atlanta      START   Verify Data Guard Broker configuration is disabled
    17 seattle      START   Verify flashback database is enabled
    18 seattle      START   Verify available flashback restore points
    19 boston       START   Verify flashback database is enabled
    20 boston       START   Verify available flashback restore points
    21 oakland      START   Verify flashback database is enabled
    22 oakland      START   Verify available flashback restore points
    23 atlanta      START   Verify flashback database is enabled
    24 atlanta      START   Verify available flashback restore points
    25 boston       START   Scan LADs for presence of atlanta destination
    26 boston       START   Test if atlanta is reachable using configured TNS service
    27 boston       START   Stop media recovery
    28 oakland      START   Stop media recovery
    29 atlanta      START   Stop media recovery
    30 boston       START   Drop guaranteed restore point DBMSRU_INITIAL
    31 boston       START   Create guaranteed restore point DBMSRU_INITIAL
    32 oakland      START   Drop guaranteed restore point DBMSRU_INITIAL
    33 oakland      START   Create guaranteed restore point DBMSRU_INITIAL
    34 atlanta      START   Drop guaranteed restore point DBMSRU_INITIAL
    35 atlanta      START   Create guaranteed restore point DBMSRU_INITIAL
    36 seattle      START   Drop guaranteed restore point DBMSRU_INITIAL
    37 seattle      START   Create guaranteed restore point DBMSRU_INITIAL

INSTID TARGET       PHASE   DESCRIPTION
------ ------------ ------- ------------------------------------------------------------
    38 boston       START   Start media recovery
    39 boston       START   Verify media recovery is running
    40 oakland      START   Start media recovery
    41 oakland      START   Verify media recovery is running
    42 atlanta      START   Start media recovery
    43 atlanta      START   Verify media recovery is running
    44 seattle      START   Verify user_dump_dest has been specified
    45 seattle      START   Backup control file to rolling_change_backup.f
    46 boston       START   Verify user_dump_dest has been specified
    47 boston       START   Backup control file to rolling_change_backup.f
    48 oakland      START   Verify user_dump_dest has been specified
```

```
     49 oakland      START   Backup control file to rolling_change_backup.f
     50 atlanta      START   Verify user_dump_dest has been specified
     51 atlanta      START   Backup control file to rolling_change_backup.f
     52 seattle      START   Get current redo branch of the primary database
     53 boston       START   Wait until recovery is active on the primary's redo branch
     54 boston       START   Stop media recovery
     55 seattle      START   Execute dbms_logstdby.build
     56 boston       START   Convert into a transient logical standby
     57 boston       START   Open database
     58 boston       START   Configure logical standby parameters
     59 boston       START   Start logical standby apply
     60 boston       START   Get redo branch of transient logical standby
     61 boston       START   Get reset scn of transient logical redo branch
     62 atlanta      START   Stop media recovery
     63 atlanta      START   Flashback database
     64 seattle      START   Disable log file archival to atlanta
     65 boston       START   Enable log file archival to atlanta
     66 boston       START   Wait for log archive destination to atlanta to reach a valid state
     67 atlanta      START   Wait until transient logical redo branch has been registered
     68 atlanta      START   Start media recovery
     69 atlanta      START   Wait until v$dataguard_stats has been initialized
     70 atlanta      START   Wait until recovery has started on the transient redo branch
     71 seattle      START   Log pre-switchover instructions to events table
     72 boston       START   Record start of user upgrade of boston
     73 boston       SWITCH  Verify database is in OPENRW mode
     74 boston       SWITCH  Record completion of user upgrade of boston

INSTID TARGET       PHASE   DESCRIPTION
------ ------------ ------- ------------------------------------------------------
     75 boston       SWITCH  Scan LADs for presence of seattle destination
     76 boston       SWITCH  Scan LADs for presence of oakland destination
     77 boston       SWITCH  Scan LADs for presence of atlanta destination
     78 boston       SWITCH  Test if seattle is reachable using configured TNS service
     79 boston       SWITCH  Test if oakland is reachable using configured TNS service
     80 boston       SWITCH  Test if atlanta is reachable using configured TNS service
     81 seattle      SWITCH  Enable log file archival to boston
     82 boston       SWITCH  Enable log file archival to atlanta
     83 boston       SWITCH  Start logical standby apply
     84 atlanta      SWITCH  Start media recovery
     85 atlanta      SWITCH  Wait until upgrade redo has been fully recovered
     86 boston       SWITCH  Wait until apply lag has fallen below 600 seconds
     87 seattle      SWITCH  Log post-switchover instructions to events table
     88 seattle      SWITCH  Switch database to a logical standby
     89 boston       SWITCH  Wait until end-of-redo has been applied
     90 oakland      SWITCH  Wait until end-of-redo has been applied
     91 seattle      SWITCH  Disable log file archival to oakland
     92 boston       SWITCH  Switch database to a primary
     93 oakland      SWITCH  Stop media recovery
     94 seattle      SWITCH  Synchronize plan with new primary
     95 seattle      FINISH  Verify only a single instance is active
     96 seattle      FINISH  Verify database is mounted
     97 seattle      FINISH  Flashback database
     98 seattle      FINISH  Convert into a physical standby
     99 oakland      FINISH  Verify database is mounted
    100 oakland      FINISH  Flashback database
    101 boston       FINISH  Verify database is open
    102 boston       FINISH  Save the DBID of the new primary
    103 boston       FINISH  Save the logminer session start scn
    104 seattle      FINISH  Wait until transient logical redo branch has been registered
    105 oakland      FINISH  Wait until transient logical redo branch has been registered
    106 seattle      FINISH  Start media recovery
```

```
   107 oakland      FINISH  Start media recovery
   108 seattle      FINISH  Wait until apply/recovery has started on the transient branch
   109 oakland      FINISH  Wait until apply/recovery has started on the transient branch
   110 seattle      FINISH  Wait until upgrade redo has been fully recovered

INSTID TARGET       PHASE   DESCRIPTION
------ ------------ ------- ------------------------------------------------
   111 oakland      FINISH  Wait until upgrade redo has been fully recovered
   112 seattle      FINISH  Drop guaranteed restore point DBMSRU_INITIAL
   113 boston       FINISH  Drop guaranteed restore point DBMSRU_INITIAL
   114 oakland      FINISH  Drop guaranteed restore point DBMSRU_INITIAL
   115 atlanta      FINISH  Drop guaranteed restore point DBMSRU_INITIAL

115 rows selected.

SQL>
```

The columns in this view display the following information:

- `INSTID` - The Instruction ID, which is the order in which the instruction is to be performed. Instructions are typically performed in groups.
- `PHASE` - Every instruction in the upgrade plan is associated with a particular phase. A phase is a logical grouping of instructions which is performed by a procedure in the `DBMS_ROLLING` PL/SQL package. When a `DBMS_ROLLING` procedure is invoked, all of the associated instructions in the upgrade plan for that phase are executed. Possible phases are as follows:
  - `START`: Consists of activities related to setup such as taking restore points, instantiation of the transient logical standby database, and configuration of LGS databases. Activities in this phase are initiated when you call the `DBMS_ROLLING.START_PLAN` procedure. See Step 1 in "Performing a Rolling Upgrade".
  - `SWITCH`: Consists of activities related to the switchover of the transient logical standby into the new primary database. Activities in this phase are initiated when you call the `DBMS_ROLLING.SWITCHOVER` procedure. See Step 3 in"Performing a Rolling Upgrade"
  - `FINISH`: Consists of activities related to configuring standby databases for recovery of the upgrade redo. Activities in this phase are initiated when you call the `DBMS_ROLLING.FINISH_PLAN` procedure. See Step 5 in "Performing a Rolling Upgrade".
- `EXEC_STATUS` - The overall status of the instruction.
- `PROGRESS` - The progress of an instruction's execution. A value of `REQUESTING` indicates an instruction is being transmitted to a target database for execution. A value of `EXECUTING` indicates the instruction is actively being executed. A value of `REPLYING` indicates completion information is being returned.
- `DESCRIPTION` - The specific operation that is scheduled to be performed.
- `TARGET` - The site at which a given instruction is to be performed.
- `EXEC_INFO` - Additional contextual information related to the instruction.

> **See Also:**
>
> - *Oracle Database Reference* for more information about the `DBA_ROLLING_PLAN` view

6. Upgrade plans need to be revised after any change to the rolling upgrade or database configuration. A configuration change could include any of the following:
   - init.ora parameter file changes at any of the databases participating in the rolling upgrade
   - database role changes as a result of failover events
   - rolling upgrade parameter changes

   To revise an active upgrade plan, call the `BUILD_PLAN` procedure again. In some cases, the `BUILD_PLAN` procedure may raise an error if a given change cannot be accepted. For example, setting the `ACTIVE_SESSIONS_WAIT` parameter has no effect if the switchover has already occurred.

   It is recommended that you call the `BUILD_PLAN` procedure to process a group of parameter changes rather than processing parameters individually.

Example 14-1 Setting Switchover to Enforce Apply Lag Requirements

The following example demonstrates how to configure the plan to wait for the apply lag to fall below 60 seconds before switching over to the future primary:

```
DBMS_ROLLING.SET_PARAMETER('SWITCH_LGM_LAG_WAIT', '1');
DBMS_ROLLING.SET_PARAMETER('SWITCH_LGM_LAG_TIME', '60');
```

Example 14-2 Resetting Logging Back to Its Default Value

The following example demonstrates resetting the `LOG_LEVEL` global parameter back to its default value.

```
DBMS_ROLLING.SET_PARAMETER (
  name=>'LOG_LEVEL',
  value=>NULL);
```

Example 14-3 Designating a Database as an Optional Participant

The following example demonstrates setting the `INVOLVEMENT` local parameter of database `atlanta` to indicate that errors encountered on the database should not impede the overall rolling upgrade.

```
DBMS_ROLLING.SET_PARAMETER (
  scope=>'atlanta',
  name=>'involvement',
  value=>'optional');
```

Example 14-4 Setting a Database to Protect the Transient Logical Standby

The following example demonstrates setting the `MEMBER` local parameter of database `atlanta` to indicate it should protect the transient logical standby database during the rolling upgrade.

```
DBMS_ROLLING.SET_PARAMETER (
  scope=>'atlanta',
```

```
name=>'member',
value=>'leading');
```

14.5 Performing a Rolling Upgrade

Follow these steps to perform a rolling upgrade using the `DBMS_ROLLING` PL/SQL package.

Table 14-2 provides a summary of the steps. These steps assume that you have first successfully built an upgrade plan as described in "Planning a Rolling Upgrade".

Table 14-2 Steps to Perform Rolling Upgrade Using DBMS_ROLLING

Step	Description	PHASE
Step 1	Call the `DBMS_ROLLING.START_PLAN` procedure to configure the future primary and physical standbys designated to protect the future primary.	START
Step 2	Manually upgrade the Oracle Database software at the future primary database and standbys that protect it.	SWITCH PENDING
Step 3	Call the `DBMS_ROLLING.SWITCHOVER` procedure to switch roles between the current and future primary database.	SWITCH
Step 4	Manually restart the former primary and remaining standby databases on the higher version of Oracle Database.	FINISH PENDING
Step 5	Call the `DBMS_ROLLING.FINISH_PLAN` procedure to convert the former primary to a physical standby, and to configure the remaining standby databases for recovery of the upgrade redo.	FINISH

Activities that take place during each step belong to a specific phase of the rolling upgrade as shown in the PHASE column of Table 14-2. A rolling upgrade operation is at a single phase at any given time. The current phase of a rolling upgrade is reported in the `PHASE` column of the `DBA_ROLLING_STATUS` view.

The rest of this section describes each of the upgrade steps in detail.

1. Call the `DBMS_ROLLING.START_PLAN` procedure to configure the future primary and physical standbys designated to protect the future primary.

   The `DBMS_ROLLING.START_PLAN` procedure is the formal start of the rolling upgrade. The goal of the `START_PLAN` procedure is to configure the transient logical standby database and any physical standby databases that have been designated to protect it. When invoked, the `START_PLAN` procedure executes all instructions in the upgrade plan with a `PHASE` value of `START_PLAN`. The types of instructions that are performed include:

   - Backing up the control file for each database to a trace file
   - Creating flashback database guaranteed restore points
   - Building a LogMiner dictionary at the primary database
   - Recovering the designated physical standby into a transient logical standby database
   - Loading the LogMiner dictionary into the logical standby database
   - Configuring LGS databases with the transient logical standby database

   Call the `START_PLAN` procedure as follows (no arguments are required):

```
SQL> EXECUTE DBMS_ROLLING.START_PLAN;
```

2. Manually upgrade the Oracle Database software at the future primary database and standbys that protect it.

   After the START_PLAN procedure has completed, you must manually upgrade the Oracle Database software at the future primary database and standbys which protect the future primary database. This involves the following steps:

   a. Upgrade the Oracle Database software of the transient logical (LGM) and leading group standbys (LGS).

   b. Start media recovery on the LGS databases.

   c. Upgrade the transient logical standby database either manually or using the Database Upgrade Assistant (DBUA).

   d. Re-open the transient logical standby in read/write mode.

   The transient logical standby and LGS databases are a functional group. The LGS databases must be restarted on the higher version actively running media recovery before the transient logical standby is upgraded. If the LGS databases are not configured first, then the upgrade of the transient logical is not protected. At the conclusion of this step, the upgrade of the transient logical is complete, and media recovery is running on all LGS databases.

   It is recommended that you wait until all LGS databases have been fully upgraded before performing the switchover. An LGS database is fully upgraded when its associated record in the DBA_ROLLING_DATABASES view reports a value of YES in the UPDATED column.

3. Call the DBMS_ROLLING.SWITCHOVER procedure to switch roles between the current and future primary database.

   The SWITCHOVER procedure switches roles between the current and future primary databases. The procedure times the switchover to occur when apply lag is minimal which minimizes outage time of the primary service. The SWITCHOVER procedure executes all instructions in the upgrade plan with a PHASE value of SWITCHOVER. The types of instructions that are performed can include:

   - Waiting for the apply lag at the Leading Group Master (LGM), which is currently the transient logical standby, to fall below a threshold value
   - Waiting for the apply lag at LGS databases to fall below a threshold value
   - Switching the primary to the logical standby role
   - Switching the Leading Group Master (LGM), which is currently a logical standby, to the primary role
   - Enabling log archive destinations at the Leading Group Master (LGM) after it has become the new primary

   Call the SWITCHOVER procedure as follows (no arguments are required):

   ```
   SQL> EXECUTE DBMS_ROLLING.SWITCHOVER;
   ```

   If a switchover error occurs after the switchover of the primary to the standby role but before the transient logical could be successfully converted into the primary role, then continue to execute the SWITCHOVER procedure at the former primary site until successful completion.

4. At this point, you must manually restart and mount the former primary and remaining standby databases on the higher version of Oracle Database. Mounting

the standby databases is especially important because the DBMS_ROLLING package needs to communicate with the standby database to continue the rolling upgrade.

5. The overall goal of the FINISH_PLAN procedure is to configure the former primary and TGP standbys as physical standbys which recover through the upgrade redo. When invoked, the FINISH_PLAN procedure executes all instructions in the upgrade plan with a PHASE value of FINISH. The types of instructions that are performed include:

   - Flashback of the former primary and TGP standbys
   - Conversion of the former primary into a physical standby
   - Startup of media recovery on the new redo branch

   Call the FINISH_PLAN procedure as follows (no arguments are required):

   ```
   SQL> EXECUTE DBMS_ROLLING.FINISH_PLAN;
   ```

14.6 Monitoring a Rolling Upgrade

There are several views you can query for information about the databases involved in a rolling upgrade.

- DBA_ROLLING_STATUS

  Provides information about the overall status of the upgrade.

- DBA_ROLLING_DATABASES

  Provides information about the role, protection, and recovery state of each database involved in the rolling upgrade.

- DBA_ROLLING_STATISTICS

  Provides statistics such as start and finish times, how long services were offline, and so on.

> **See Also:**
>
> - *Oracle Database Reference* for descriptions of these views

14.7 Rolling Back a Rolling Upgrade

To roll back a rolling upgrade procedure, you can call the DBMS_ROLLING.ROLLBACK_PLAN procedure.

The procedure is called as follows:

```
DBMS_ROLLING.ROLLBACK_PLAN;
```

The ROLLBACK_PLAN procedure has the following requirements:

- The ROLLBACK_PLAN procedure can only be called if the DBMS_ROLLING.SWITCHOVER procedure has not been previously called.

- Before you can use the `ROLLBACK_PLAN` procedure you must set the transient logical standby database back to a mounted state because a flashback database is imminent.
- If the Oracle Database software was already upgraded, then you must restart the resultant physical standbys on the older version, and start media recovery.

14.8 Handling Role Changes That Occur During a Rolling Upgrade

If a situation arises in which a rolling upgrade is underway and you need to perform a failover in your Oracle Data Guard configuration before the rollover completes, you can do so only in these circumstances.

- The failover was not performed while a `DBMS_ROLLING` procedure was in progress.
- The failover was between a primary database and a physical standby database, and was a no-data-loss failover.
- The failover was between a transient logical standby database and a physical standby of the transient logical standby database.

A role change is a significant event that inevitably invalidates instructions in the upgrade plan, which was tailored for a different configuration. To resume the rolling upgrade, a new plan must be created. You must set the `FAILOVER` parameter to indicate that the configuration has changed. This parameter is detected on the next invocation of the `BUILD_PLAN` procedure, and the existing plan is amended accordingly.

After the revised plan is built, you can resume the rolling upgrade.

14.9 Examples of Rolling Upgrades

These examples show a variety of rolling upgrade scenarios.

At some point in all of the scenarios, the same basic rolling upgrade steps are used. These steps are shown in Example 14-5. The rest of the examples refer back to this example where appropriate rather than reiterating the same steps.

Some of the examples in this section instruct you to resume the rolling upgrade, which means that you should continue where you left off. Resuming a rolling upgrade involves identifying the current phase of the rolling upgrade and reperforming either the PL/SQL procedure associated with the phase or the activities relevant to the phase. The current phase of the rolling upgrade is shown in the `PHASE` column of the `DBA_ROLLING_STATUS` view.

> **Note:**
>
> The scenarios provided in this section are only meant to be hypothetical examples. You can use the Rolling Upgrade Using Oracle Active Data Guard feature to perform database upgrades beginning with the first Oracle Database 12*c* patchset.

Example 14-5 Basic Rolling Upgrade Steps

1. Start the rolling upgrade:

```
SQL> EXECUTE DBMS_ROLLING.START_PLAN;
```

2. Upgrade the transient logical standby and its protecting standbys.
   a. Mount LGP standbys using the higher Oracle Database software version.
   b. Start media recovery on Leading Group Physicals (LGP).
   c. Open the Leading Group Master (LGM), which is the transient logical standby, in upgrade mode using the higher Oracle Database software version.
   d. Upgrade the Leading Group Master (LGM), which is the transient logical standby, either manually or using the Database Upgrade Assistant (DBUA).
   e. Restart the Leading Group Master (LGM), which is the transient logical standby, in read/write mode.
3. Switchover to the Leading Group Master (LGM):

```
SQL> EXECUTE DBMS_ROLLING.SWITCHOVER;
```

4. Restart the databases in the trailing group. This includes the original primary database and all its protecting standbys in the trailing group (TGP).
   a. Mount the former primary using the higher Oracle Database version.
   b. Mount the physical standbys of the former primary using the higher Oracle Database version.
5. Finish the rolling upgrade:

```
SQL> EXECUTE DBMS_ROLLING.FINISH_PLAN;
```

Example 14-6 Rolling Upgrade Between Two Databases

The following example demonstrates a rolling upgrade on a two-site configuration consisting of a primary database and a physical standby database. In this example, `seattle` is the current primary and `boston` is the future primary. The `seattle` database is automatically chosen as the Trailing Group Master (TGM) and participates in the operation. By default, there is nothing that needs to be set for `seattle`.

1. Initialize the upgrade plan:

```
SQL> EXECUTE DBMS_ROLLING.INIT_PLAN(future_primary=>'boston');
```

2. Build the upgrade plan:

```
SQL> EXECUTE DBMS_ROLLING.BUILD_PLAN;
```

3. Perform the rolling upgrade as described in Example 14-5.

Example 14-7 Rolling Upgrade Between Three Databases

The following example demonstrates a rolling upgrade on a three-site configuration consisting of a primary databases and two physical standby databases. In this example, `seattle` is the primary, `boston` is the future primary, and `oakland` is a physical standby of `seattle`.

1. Initialize the upgrade plan:

```
SQL> EXECUTE DBMS_ROLLING.INIT_PLAN (future_primary => 'boston');
```

2. Build the upgrade plan:

```
SQL> EXECUTE DBMS_ROLLING.BUILD_PLAN;
```

3. Perform the rolling upgrade as described in Example 14-5.

Example 14-8 Rolling Upgrade Between Four Databases

The following example demonstrates a rolling upgrade on a four-site configuration consisting of a primary database and three physical standby databases. In this example, `seattle` is the primary database, `boston` is the future primary, `oakland` is a physical standby of `seattle`, and `atlanta` is a physical standby of `boston`.

1. Initialize the upgrade plan:

```
SQL> EXECUTE DBMS_ROLLING.INIT_PLAN (future_primary => 'boston');
```

2. Configure `atlanta` as a standby in the leading group:

```
SQL> EXECUTE DBMS_ROLLING.SET_PARAMETER(scope=>'atlanta',name=>'member',
 value=>'leading');
```

3. Build the upgrade plan:

```
SQL> EXECUTE DBMS_ROLLING.BUILD_PLAN;
```

4. Perform the rolling upgrade as described in Example 14-5.

Example 14-9 Rolling Upgrade on a Reader Farm

The following example demonstrates a rolling upgrade on a reader farm configuration consisting of one primary database and nine physical standby databases. In this example, eight physical standby databases are split into two groups of four in order for physical standbys to be available as Oracle Active Data Guard standbys before and after the switchover. In this example, `seattle` is the primary, `boston` is the future primary, databases `rf[a-d]` are physical standbys of `seattle`, and databases `rf[e-h]` are physical standbys of `boston`. The rolling upgrade is configured so that the switchover to the new primary waits until the apply lag among the reader farm group of the future primary database is less than 60 seconds.

1. Initialize the upgrade plan:

```
SQL> EXECUTE DBMS_ROLLING.INIT_PLAN ( future_primary => 'boston');
```

2. Configure the reader farm group to protect the future primary:

```
SQL> EXECUTE DBMS_ROLLING.SET_PARAMETER(scope=>'rfe',name=>'member',
value=>'leading');

SQL> EXECUTE DBMS_ROLLING.SET_PARAMETER(scope=>'rff',name=>'member',
value=>'leading');

SQL> EXECUTE DBMS_ROLLING.SET_PARAMETER(scope=>'rfg',name=>'member',
value=>'leading');

SQL> EXECUTE DBMS_ROLLING.SET_PARAMETER(scope=>'rfh',name=>'member',
value=>'leading');
```

3. Set a maximum permitted apply lag of 60 seconds on the future primary's reader farm:

```
SQL> EXECUTE DBMS_ROLLING.SET_PARAMETER(name=>'SWITCH_LGS_LAG_WAIT',
 value=>'1');
```

4. Build the upgrade plan:

```
SQL> EXECUTE DBMS_ROLLING.BUILD_PLAN;
```

5. Perform the rolling upgrade as described in Example 14-5.

Example 14-10 Rolling Upgrade for Application Testing

The following example demonstrates using rolling upgrade on a four-site configuration to configure a transient logical standby and a physical of the transient logical standby in order to validate an application on the higher version database. The primary database is `seattle`, `boston` is the future primary, `oakland` is a physical standby of `seattle`, and `atlanta` is physical standby of `boston`. So in this example, `seattle` and `oakland` make up the trailing group, and `boston` and `atlanta` make up the leading group. At the end of testing, `boston` and `atlanta` are restored back to their original physical standby roles in order to resume protection of `seattle`.

1. Initialize the upgrade plan:

```
SQL> EXECUTE DBMS_ROLLING.INIT_PLAN (future_primary => 'boston');
```

2. Configure `atlanta` to protect the future primary:

```
SQL> EXECUTE DBMS_ROLLING.SET_PARAMETER(scope=>'atlanta',name=>'member',
 value=>'leading');
```

3. Build the upgrade plan:

```
SQL> EXECUTE DBMS_ROLLING.BUILD_PLAN;
```

4. Start the rolling upgrade:

```
SQL> EXECUTE DBMS_ROLLING.START_PLAN;
```

5. Upgrade `boston` and `atlanta`:
   a. Mount `atlanta` using the higher database version.
   b. Start media recovery on `atlanta`.
   c. Open `boston` in upgrade mode using the higher database version.
   d. Upgrade database `boston` either manually or using the Database Upgrade Assistant (DBUA).
   e. Restart `boston` in read/write mode.
6. Test the application, as necessary.
7. Rollback the configuration:
   a. Restart `boston` in mounted mode
   b. Roll back the upgrade:

```
SQL> EXECUTE DBMS_ROLLING.ROLLBACK_PLAN;
```

8. Start media recovery on `boston` and `atlanta` using the older database version:
   a. Mount `boston` and `atlanta` using the lower database version.
   b. Start media recovery on `boston` and `atlanta`.

Example 14-11 Resuming a Rolling Upgrade After a Failover to a New Primary

The following example demonstrates a no-data-loss failover of a physical standby to the primary role followed by the reconfiguration of the rolling upgrade plan on a three-site configuration. In this example, `seattle` is the primary, `boston` is the future primary, and `oakland` is a physical standby of `seattle`. Database `oakland` is failed over to become the new primary. (The Trailing Group is (`seattle`, `oakland`) and the Leading Group is `boston`.)

1. Recover remaining redo on `oakland`, and fail over to the new primary role:

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

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

SQL> STARTUP OPEN;
```

2. Configure log archive destinations on `oakland`, as necessary:

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_2='service="boston" reopen=5
  2  LGWR ASYNC NET_TIMEOUT=180 valid_for=(ONLINE_LOGFILE, PRIMARY_ROLE)
  3  DB_UNIQUE_NAME="oakland"';
```

3. Set a parameter to indicate that a failover took place:

```
SQL> EXECUTE DBMS_ROLLING.SET_PARAMETER(name=>'failover', value=>'1');
```

4. Revise the upgrade plan:

```
SQL> EXECUTE DBMS_ROLLING.BUILD_PLAN;
```

5. Resume the rolling upgrade.

Example 14-12 Resuming a Rolling Upgrade After a Failover to a New Transient Logical

The following example demonstrates a failover of a physical standby to the transient logical role, followed by the reconfiguration of the rolling upgrade plan on a five-site configuration. In this example, `seattle` is the primary, `boston` is the future primary, `oakland` is a physical standby of `seattle`, and `atlanta` and `miami` are physical standbys of `boston`. Database `atlanta` is failed over to become the new transient logical standby.

1. Recover remaining redo on `atlanta` and failover to the new transient logical role:

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

SQL> ALTER DATABASE ACTIVATE STANDBY DATABASE;

SQL> ALTER DATABASE OPEN;
```

2. Configure log archive destinations on `atlanta`, as necessary:

```
SQL> alter system set log_archive_dest_2='service="seattle" reopen=5
  2  LGWR ASYNC NET_TIMEOUT=180 valid_for=(ONLINE_LOGFILE, PRIMARY_ROLE)
  3  DB_UNIQUE_NAME="atlanta"';

SQL> alter system set log_archive_dest_3='service="oakland" reopen=5
  2  LGWR ASYNC NET_TIMEOUT=180 valid_for=(ONLINE_LOGFILE, PRIMARY_ROLE)
  3  DB_UNIQUE_NAME="atlanta"';

SQL> alter system set log_archive_dest_3='service="miami" reopen=5
  2  LGWR ASYNC NET_TIMEOUT=180 valid_for=(ONLINE_LOGFILE, ALL_ROLES)
  3  DB_UNIQUE_NAME="atlanta"';
```

3. Specify `atlanta` as the new transient logical standby database:

```
SQL> EXECUTE DBMS_ROLLING.SET_PARAMETER(name=>'failover', value=>'1');
```

4. Revise the upgrade plan:

```
SQL> EXECUTE DBMS_ROLLING.BUILD_PLAN;
```

5. Resume the rolling upgrade.

15
Oracle Data Guard Scenarios

These scenarios present different situations you might encounter while administering your Oracle Data Guard configuration. Each of them can be adapted to your specific environment.

15.1 Configuring Logical Standby Databases After a Failover

These are 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:

15.1.1 When the New Primary Database Was Formerly a Physical Standby Database

These steps demonstrate 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.

1. On the SAT database, issue the following statement to configure the FAL_SERVER parameter to enable automatic recovery of log files.

```
SQL> ALTER SYSTEM SET FAL_SERVER='<tns_name_to_new_primary>';
```

2. Call the PREPARE_FOR_NEW_PRIMARY routine to verify that the logical standby database is capable of serving as a standby database to the new primary database. During this step, local copies of log files that pose a risk for data divergence are deleted from the local database. These log files are then requested for re-archival directly from 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:
>
> a. Flash back the database to the SCN as stated in the warning and then
>
> b. Repeat this step before continuing.
>
> See Flashing Back a Logical Standby Database to a Specific Applied SCN for an example of how to flash back a logical standby database to an Apply SCN.

3. On the SAT database, issue the following statement to start SQL Apply:

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

> **Note:**
>
> A value of `NONE` must be returned before you attempt to reinstate an old primary database.

15.1.2 When the New Primary Database Was Formerly a Logical Standby Database

These steps demonstrate 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.

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 `NONE`. Otherwise the new primary database has not completed the work required to enable support for logical standby databases. For example:

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

A value of NONE must be returned before you attempt to reinstate an old primary database.

2. On the SAT database, issue the following statement to configure the FAL_SERVER parameter to enable automatic recover of log files:

```
SQL> ALTER SYSTEM SET FAL_SERVER='<tns_name_to_new_primary>';
```

3. Call the PREPARE_FOR_NEW_PRIMARY routine to verify the logical standby database is capable of being a standby to the new primary. During this step, local copies of log files which pose a risk for data divergence are deleted from the local database. These log files are then requested for re-archival directly from the new primary database.

   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:
>
> a. Flash back the database to the SCN as stated in the warning and then
>
> b. Repeat this step before continuing.
>
> See Flashing Back a Logical Standby Database to a Specific Applied SCN for an example of how to flash back a logical standby database to an Apply SCN.

4. On the SAT database, issue the following statements to start SQL Apply:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY NEW PRIMARY nyc_link;
```

   This statement must always be issued without the real-time apply option enabled. 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;
```

15.2 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 Oracle 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 information about automatic reinstatement of the failed primary database as a new standby database (as an alternative to using Flashback Database)

15.2.1 Flashing Back a Failed Primary Database into a Physical Standby Database

These steps bring the old primary database back into the Oracle Data Guard configuration as 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.

1. 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;
   ```

2. 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 the previous step.

   ```
   SQL> SHUTDOWN IMMEDIATE;
   SQL> STARTUP MOUNT;
   SQL> FLASHBACK DATABASE TO SCN standby_became_primary_scn;
   ```

3. To convert the database to a physical standby database, issue the following statement on the old primary database:

   ```
   SQL> ALTER DATABASE CONVERT TO PHYSICAL STANDBY;
   ```

4. To start transporting redo to the new physical standby database, perform the following steps on the new primary database:

   a. 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 FROM V$ARCHIVE_DEST_STATUS;
      ```

   b. If necessary, enable the destination:

      ```
      SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_n=ENABLE;
      ```

c. 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 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.

5. Start Redo Apply on the new physical standby database:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE 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 needs 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 "Performing a Switchover to a Physical Standby Database" for more information.

15.2.2 Flashing Back a Failed Primary Database into a Logical Standby Database

These steps bring the old primary database back into the Oracle Data Guard configuration as a new logical standby database without having to formally instantiate it from the new primary database.

These steps assume that the Oracle 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.

1. Determine the flashback SCN and the recovery SCN. The flashback SCN is the SCN to which the failed primary database is flashed back. The recovery SCN is the SCN to which the failed primary database is 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 -
> FROM DBA_LOGSTDBY_HISTORY -
> WHERE stream_sequence# = (SELECT MAX(stream_sequence#)-1 -
> FROM DBA_LOGSTDBY_HISTORY);
```

2. Flash back the failed primary database to the flashback SCN identified in the previous step:

```
SQL> FLASHBACK DATABASE TO SCN flashback_scn;
```

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;
```

4. Configure the `FAL_SERVER` parameter to enable automatic recovery of log files. Issue the following statement on the physical standby (failed primary):

```
SQL> ALTER SYSTEM SET FAL_SERVER='<tns_name_to_new_primary>';
```

5. 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 fast recovery area. You can use the following RMAN command to delete the relevant archive logs from the fast recovery area:

   ```
   RMAN> DELETE FORCE ARCHIVELOG FROM SCN ARCHIVE_SCN;
   ```

   Once deleted, these divergent logs and subsequent transactions can never be recovered.

6. Recover until the recovery SCN identified in Step 1:

   ```
   SQL> RECOVER MANAGED STANDBY DATABASE UNTIL CHANGE recovery_scn;
   ```

7. Enable the database guard:

   ```
   SQL> ALTER DATABASE GUARD ALL;
   ```

8. Activate the physical standby to become a primary database:

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

9. Open the database:

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

10. Create a database link to the new primary, and start SQL Apply:

    ```
    SQL> CREATE PUBLIC DATABASE LINK mylink -
    > CONNECT TO system IDENTIFIED BY password -
    > USING 'service_name_of_new_primary_database';

    SQL> ALTER DATABASE START LOGICAL STANDBY APPLY NEW PRIMARY mylink;
    ```

    The role reversal is now complete.

15.2.3 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. These steps describe how to use Flashback Database and SQL Apply to recover to a known applied SCN.

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) -
   > AS TARGET_SCN FROM DUAL;
   ```

2. 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;
```

3. Issue the following query to confirm SQL Apply has applied less than or up to the APPLIED_SCN.

```
SQL> SELECT APPLIED_SCN FROM V$LOGSTDBY_PROGRESS;
```

15.3 Using Flashback Database After Issuing an Open Resetlogs Statement

If an error occurrs on the primary database in an Oracle Data Guard configuration in which the standby database is using real-time apply, then the same error is applied on the standby database. if Flashback Database is enabled, you can revert the primary and standby databases back to their pre-error condition.

To do so, issue 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 Creating a Physical Standby Database and Creating a Logical Standby Database , after the point-in-time recovery was performed on the primary database.)

15.3.1 Flashing Back a Physical Standby Database to a Specific Point-in-Time

These steps describe how to avoid re-creating a physical standby database after you issue the OPEN RESETLOGS statement on the primary database.

1. Determine the SCN before the RESETLOGS operation occurred. For example, 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;
```

2. On the standby database, obtain the current SCN with the following query:

```
SQL> SELECT TO_CHAR(CURRENT_SCN) FROM V$DATABASE;
```

3. 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, and the new branch of redo from the OPEN RESETLOGS statement has been registered at the standby, then apply services can 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.

4. To start Redo Apply on the physical standby database, issue the following statement:

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

The standby database is now ready to receive and apply redo from the primary database.

15.3.2 Flashing Back a Logical Standby Database to a Specific Point-in-Time

These 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.

1. 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;
```

2. Determine the target SCN for flashback operation at the logical standby.

   In this step, the `FLASHBACK_SCN` value for `PRIMARY_SCN` is from Step 1.

```
SQL> SELECT DBMS_LOGSTDBY.MAP_PRIMARY_SCN (PRIMARY_SCN => FLASHBACK_SCN) -
> AS TARGET_SCN FROM DUAL;
```

3. 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;
```

4. Confirm that a log file from the primary's new branch is registered before SQL apply is started. Issue the following query on the primary database:

```
SQL> SELECT resetlogs_id FROM V$DATABASE_INCARNATION WHERE status = 'CURRENT';
```

   Issue the following query on the standby database:

```
SQL> SELECT * FROM DBA_LOGSTDBY_LOG WHERE resetlogs_id = resetlogs_id_at_primary;
```

If one or more rows are returned, it confirms that there are registered logfiles from the primary's new branch.

5. Start SQL Apply:

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

15.4 Recovering After the NOLOGGING Clause Is Specified

Some SQL statements allow you to specify a `NOLOGGING` clause so that the operation is not logged in the online redo log file. In actuality, a redo record is still written to the online redo log file, but there is no data associated with the 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. To avoid these problems, Oracle recommends that you always specify the `FORCE LOGGING` clause in the `CREATE DATABASE` or `ALTER DATABASE` statements.
If the database `FORCE LOGGING` clause is not enabled, then the interaction between logging and nologging clauses, specified at various different levels, is captured in the `V$DATABASE.FORCE_LOGGING` column (for CDBs) or the `DBA_PDBS.FORCE_LOGGING` column (for PDBs).

See *Oracle Database Administrator's Guide*.

15.4.1 Recovery Steps for Logical Standby Databases

On logical standby databases, when SQL Apply encounters a redo record for an operation performed with the `NOLOGGING` clause on a table that is not being skipped, 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 Adding or Re-Creating Tables On a Logical Standby Database.

> **Note:**
>
> In general, use of the `NOLOGGING` clause is not recommended. Therefore, if you know in advance that operations using the `NOLOGGING` clause will be performed on certain tables in the primary database, then you might want to prevent the application of SQL statements associated with these tables to the logical standby database. You can do this by using the `DBMS_LOGSTDBY.SKIP` procedure.

15.4.2 Recovery Steps for Physical Standby Databases

When the redo is applied to the physical standby database, a portion of the data file 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 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 use Recovery Manager (RMAN) to perform a RECOVER ... NONLOGGED BLOCK command. The following steps describe a simple approach that recovers all nonlogged blocks. (See follow-on sections for other approaches such as determining whether a backup is required after unrecoverable operations, and recovering parts of a physical standby database.)

1. Stop recovery on the standby:

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

2. Recover the nonlogged blocks by connecting RMAN to the standby and issue the following command:

   ```
   RMAN> RECOVER DATABASE NONLOGGED BLOCK;
   ```

   If the presence of unrecoverable blocks is only found after a switchover, then you can use these same two steps, but the primary database must be just mounted (not open) and, RMAN must be connected to the primary.

   It is possible that the RMAN RECOVER command may not be able to recover all the nonlogged blocks. The reasons that this might happen are detailed in the alert log of the database from which the RMAN command was executed. The most common reason is that a block has been modified recently at the primary and not yet written to its corresponding data file. This may mean that the block sent to the standby is too old to replace the unrecoverable block on the standby. To resolve this, issue the RECOVER command again at a later time after the block has been written out to the data file.

The following is an example of an alert log entry for an execution of the RECOVER command which left some blocks unrecovered:

```
Started Nonlogged Block Replacement recovery on file 7 (ospid 13005 rcvid
11319003446180375696)
Finished Nonlogged Block Replacement recovery on file 7. 5 blocks remain
  Statistics for replacement block source database (service=dg3tns)
  (Use of it stopped due to error 12942 received from it)
  Blocks requested 5, blocks received 0.

  Reason replacement blocks accepted or rejected              Blocks Last block
  ------------------------------------------------------ ---------- ----------
  Not received: Rejected by sender. Wrong state or SCN             5         21
```

In this case, the command was run on a standby and the primary did not send any blocks but instead reported the following Oracle error: .

```
ORA-12942: database incarnation at source does not match
```

An examination of the alert logs would reveal that the primary had performed FLASHBACK DATABASE and OPEN RESETLOGS commands, but the standby had not been flashed back. This means the standby would now be on an orphaned branch of redo and therefore the primary could not supply data blocks that would be known to be the correct version.

> **Note:**
>
> When a Data Guard configuration has more than one standby, an RMAN `RECOVER` command that is run at a standby attempts to fetch blocks from the primary only. When the `RECOVER` command is run at the primary it attempts to fetch blocks from only the one standby that it determines to be most likely to yield good blocks, which typically means the standby that is closest to being in synch with the primary.

15.4.3 Determining If a Backup Is Required After Unrecoverable Operations

If you performed unrecoverable operations on your primary database, then you need to determine if a new backup operation is required.

To do so, take the following 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:

```
SQL> 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 data file that is more recent than the time when the data file was last backed up, then make another backup of the data file in question.

See *Oracle Database Reference* for more information about the `V$DATAFILE` view.

15.4.4 Recovery Steps for Part of a Physical Standby Database

The RMAN `RECOVER ... NONLOGGED BLOCK` command can be used to recover blocks that belong to a set of data files or a set of tablespaces or just a single pluggable database (PDB) as well as the multitenant container database (CDB).

This ability may be useful if it is acceptable to have nonlogged blocks remain in some data files, for example because it is explicitly known that the nonlogged blocks were created as a result of loading rows into an object that has now been dropped.

The view `V$NONLOGGED_BLOCK` usually lists the ranges of known invalid blocks for each data file and the entries are maintained as part of media recovery. However, there are times when the information is not complete. Typically this is after upgrading from a release prior to Oracle Database 12*c* Release 1 (12.1) or after restoring an operating system backup of a data file. The next time media recovery is run, the stale entries are removed and any newly invalidated blocks are recorded but any prior invalid blocks do not have entries in `V$NONLOGGED_BLOCK`. The `FIRST_NONLOGGED_SCN` column in the `V$DATAFILE` view can still be used to see that there is at least one invalid block in a data file even when there are no `V$NONLOGGED_BLOCK` entries for a data file.

The RMAN command `VALIDATE ... NONLOGGED BLOCK` can be used to bring the entries in `V$NONLOGGED_BLOCK` back into synchronization with the data files. It does this by determining if the existing ranges are complete and if not, it scans the necessary data files to identify any invalid blocks and make sure they are captured by an entry in `V$NONLOGGED_BLOCK`. The `VALIDATE ... NONLOGGED BLOCK` command has the same options as the `RECOVER ... NONLOGGED BLOCK` command to validate just a set of data files or a set of tablespaces or a PDB, as well as the CDB.

> **Note:**
>
> Entries in the `V$NONLOGGED_BLOCK` view represent a superset of the invalid blocks, and some normal blocks that are close to an invalid block may be included. For example, if there is an entry for file 7 that starts at block 100 and has 50 blocks in it, then none, some, or all of those 50 blocks are invalid. After the `VALIDATE` command is run, there are no ranges that have no invalid blocks in them and the first and last block of a range are also be invalid blocks. However, there are limits to the total number of ranges that can be held in the control file so sometimes it may be necessary to merge ranges for the same file, causing regular blocks to be included in a range.

If only offline data files are to be validated or recovered then the database to which they belong can be open at the time the RMAN command is run.

15.5 Creating a Standby Database That Uses OMF or Oracle ASM

When you create standby databases, there are additional steps that must be performed if the primary database uses Oracle Managed Files (OMF) or Oracle Automatic Storage Management (Oracle ASM).

The discussion in this section is presented at a level of detail that assumes you already know how to create a physical standby database and are an experienced user of the RMAN, OMF, and Oracle ASM features.

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.

> **Note:**
>
> If OMF parameters are set on the standby, then new files on that standby are always created as OMF, regardless of how they were created on the primary. Therefore, if both the `DB_FILE_NAME_CONVERT` and `DB_CREATE_FILE_DEST` parameters are set on the standby, the `DB_CREATE_FILE_DEST` parameter takes precedence.

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 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 Oracle ASM, create an Oracle 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 data files, archived log files, and a standby control file.
3. Use the RMAN `DUPLICATE FOR STANDBY` command to copy the data files, 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 Network File Storage (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 Verify the Physical Standby Database Is Performing Properly, to verify the configuration of the physical standby database.

To create a logical standby database, continue with the standby database creation process described in Creating a Logical Standby Database , 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 Open the Logical Standby Database to start, recover, and verify the logical standby database.

> **See Also:**
>
> - Creating a Physical Standby Database
> - Creating a Logical Standby Database
> - Creating a Standby Database with Recovery Manager
> - *Oracle Database Administrator's Guide* for information about OMF
> - *Oracle Automatic Storage Management Administrator's Guide* for more information about Oracle ASM
> - *Oracle Database Backup and Recovery User's Guide* for information about RMAN

15.6 Recovering From Lost-Write Errors on a Primary Database

During media recovery in an Oracle 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.

In such a situation, if lost write detection (set with the `DB_LOST_WRITE_PROTECT` initialization parameter) is enabled at both the primary and standby, then a recovery attempt at the standby results in an `ORA-752` error. If lost write detection is not enabled, then a recovery attempt results in an `ORA-600 [3020]` error. However, not all `ORA-600 [3020]` errors are due to lost writes at the primary. Therefore, before following the guidelines given in this section, work with your Oracle Support representative to determine whether the root cause for the `ORA-600 [3020]` error was indeed a lost write that occurred on the primary. Also see "Resolving ORA-752 or ORA-600 [3020] During Standby Recovery" in the My Oracle Support Note 1265884.1 at `http://support.oracle.com`.

> **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 data files 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 data files 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 the SCN printed in the block error messages is 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 Oracle Data Guard configuration, and all other standby databases 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. You can do this using the database backup taken at the new primary in step 3. (You cannot use flashback database or the Oracle 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 has the same data files as the old standby. Therefore, any undetected lost writes that the old standby had before it was activated are not detected by the new standby, since the new standby compares the same blocks. Any new lost writes that happen on either the primary or the standby are detected.

> **See Also:**
>
> *Oracle Database Backup and Recovery User's Guide* for more information about enabling lost-write detection

15.7 Using the DBCOMP Procedure to Detect Lost Writes and Other Inconsistencies

You can use the PL/SQL procedure, `DBMS_DBCOMP.DBCOMP`, to detect lost writes and also to detect inconsistencies between a primary database and physical standby databases.

The `DBMS_DBCOMP.DBCOMP` procedure compares the same data blocks on the primary and physical standby databases. The procedure can be executed at any time. (It does not require that the `DB_LOST_WRITE_PROTECT` initialization parameter be enabled.)

You can monitor the progress of an on-going block comparison operation by querying the `V$SESSION_LONGOPS` view.

> **Note:**
>
> Logical standby databases, far sync instances, and cascaded standbys cannot be the target database for the `DBMS_DBCOMP.DBCOMP` procedure.

The `DBMS_DBCOMP.DBCOMP` procedure assumes that there is one primary database and one or more physical standby databases. The databases should be at least mounted before block comparison.

In the following example situations, assume that there is a primary database with a unique name of `dgmain`, and that physical standby databases are named `dgmainb`, `dgmainc`, `dgmaind`, and so on.

Example 15-1 Primary and All Standbys Are Mounted or Open and DBCOMP Is Executed From the Primary

In this situation, when the `DBCOMP` procedure is executed from the primary database, the specified data files are compared block by block between the primary and every physical standby database. For example, suppose that you perform the following query:

```
SQL> exec sys.dbms_dbcomp.dbcomp('1','BlockCompare',:retval);
```

The generated output files are `BlockCompare_dgmainb_1` and `BlockCompare_dgmainc_d_1`.

Example 15-2 Primary and All Standbys Are Mounted or Open and DBCOMP Is Executed From a Standby

In this situation, when the `DBCOMP` procedure is executed from one of the standby databases (for example, `dgmainb`), the specified data files are compared only between the primary and that particular standby database. Other standby databases are not considered. For example, suppose that you perform the following query:

```
SQL> exec sys.dbms_dbcomp.dbcomp('1','BlockCompare',:retval);
```

The generated output file is `BlockCompare_dgmain_1`.

Example 15-3 Primary Is Mounted or Open, But Not All Standbys Are, and DBCOMP is Executed From the Primary

In this situation, when the`DBCOMP` procedure is executed on the primary, the specified data files are compared between the primary database and the mounted or open physical standby databases. For those standby databases that are neither mounted nor open, no action is taken.

Example 15-4 Primary Is Mounted or Open, But Not All Standbys Are, and DBCOMP is Executed From a Standby

In this situation, the specified data files are compared between the primary and the standby from which the `DBCOMP` procedure is executed.

Example 15-5 Primary is Not Mounted, But Multiple Standbys Are Mounted or Open

Because the primary database is neither mounted nor open, the`DBCOMP` procedure cannot find an appropriate pair of primary and physical standby databases to compare. An `ORA` error message is not returned, but a message similar to the following is printed out in the corresponding output file: `Remote database is not in the primary role`.

Example 15-6 Primary Is Mounted or Open, But No Standbys Are Mounted or Open

Because no appropriate pair of primary and physical standby databases are found, a message is printed out in the corresponding output file, but no `ORA` error is returned.

Related Topics:

- *Oracle Database PL/SQL Packages and Types Reference*

15.8 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 one of these procedures, as appropriate.

- Flashing Back a Failed Primary Database into a Physical Standby Database
- Flashing Back a Failed Primary Database into a Logical Standby Database

The procedures in these 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 topics:

- Converting a Failed Primary into a Physical Standby Using RMAN Backups
- Converting a Failed Primary into a Logical Standby Using RMAN Backups

15.8.1 Converting a Failed Primary into a Physical Standby Using RMAN Backups

These steps 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.

1. 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;
   ```

2. 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 USING 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 data files, flashing back to an SCN is equivalent to recovering up until that SCN plus one.

3. Perform the following steps on the old primary database:

   a. Issue the following statement on the old primary database:

      ```
      SQL> ALTER DATABASE CONVERT TO PHYSICAL STANDBY;
      ```

   b. Shut down and restart the database:

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

4. 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 data files during the failover.

5. If you have purchased a license for the Oracle Active Data Guard option and would like to operate your physical standby database in active query mode, skip this step. Otherwise, bring your standby database to the mount state.

   For example:

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

6. 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:

   a. 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 FROM V$ARCHIVE_DEST_STATUS;
      ```

   b. If necessary, enable the destination:

      ```
      SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_n=ENABLE;
      ```

   c. 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 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 Oracle Data Guard configuration operates properly after a role transition.

7. Start Redo Apply on the new physical standby database, as follows:

   ```
   SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE 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 "Performing a Switchover to a Physical Standby Database" for more information.

15.8.2 Converting a Failed Primary into a Logical Standby Using RMAN Backups

These steps describe how to convert a failed primary into a logical standby using RMAN backups.

1. 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:

   a. Ensure all standby redo logs have been archived. Issue the following query, looking for a value of `NONE` 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 `NONE` is returned.

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

   b. After a status of `NONE` 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 -
      > WHERE NAME='STANDBY_BECAME_PRIMARY_SCN';
      ```

2. 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 fast recovery area. You can use the following RMAN command to delete the relevant archive logs from the fast recovery area:

   ```
   RMAN> DELETE ARCHIVELOG FROM SCN ARCHIVE_SCN;
   ```

   Once deleted, these divergent logs and subsequent transactions can never be recovered.

3. 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 fast 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;
```

4. 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.

   a. 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.

   b. Use the backup to restore the data files of the failed primary database.

   c. Turn off flashback database, if it is enabled (necessary for the `USING BACKUP CONTROLFILE` clause).

   d. 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;
   ```

5. Open the database with the RESETLOGS option:

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

6. Enable Database Guard

   ```
   SQL> ALTER DATABASE GUARD ALL;
   ```

7. Create a database link to the new primary database and start SQL Apply:

   ```
   SQL> CREATE PUBLIC DATABASE LINK myLink -
   > CONNECT TO SYSTEM IDENTIFIED BY password -
   > 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 4c, let this restart turn off `RESTRICTED SESSION`.

15.9 Changing the Character Set of a Primary Without Re-Creating Physical Standbys

Oracle Data Guard allows you to change both the database character set and the national character set of a primary database without requiring you to recreate any physical standby databases in the configuration.

You can continue to use your physical standby database with minimal disruption while performing character set conversion of a primary database.

The character set migration process consists of preparatory steps such as scanning for possible issues and identifying methods to solve them. During the execution of these preparatory steps the Oracle Data Guard configuration can operate unchanged and no extra steps are required to maintain the physical standby. After the preparatory steps are completed, the actual conversion is performed which may involve changes

to both system data (metadata) and user data. Several procedures specific to Oracle Data Guard must be run as part of the conversion. The steps to run these procedures are interspersed with the steps performed by the Database Migration Assistant for Unicode (DMU) or other appropriate character set migration tool.

For a detailed description of the steps involved in this process, see My Oracle Support note 1124165.1 at http://support.oracle.com.

15.10 Actions Needed On a Standby After a PDB PITR or PDB Flashback On a Primary

After you perform a PDB PITR or PDB Flashback on a primary, you can either restore the PDB or flashback the PDB on the standby to let the standby follow the primary.

When a PDB PITR or PDB flashback is performed on the primary, and redo for the start of the operation is encountered for the first time, the MRP at the standby terminates with error ORA-39874, followed by the supplemental error ORA-39873. The following is an example of the messages that may appear in the alert log:

```
Recovery of pluggable database PDB1 aborted due to pluggable database open
resetlog marker.
To continue recovery, restore all data files for this PDB to
checkpoint SCN lower than 1437261, or timestamp before 11/15/2012 16:38:49,
and restart recovery
MRP0: Background Media Recovery terminated with error 39874

ORA-39874: Pluggable Database PDB1 recovery halted
ORA-39873: Restore all data files to a checkpoint SCN lower than 1437261.
```

Before media recovery on the standby can continue any further, you must restore all data files for that PDB. You can do this in two ways:

- If flashback is enabled on the standby, then you can use PDB flashback on the standby and then restart standby managed recover. See “Performing Recovery When Flashback is Enabled” below.
- If flashback in not enabled on the standby, then recovery can be done from a backup taken at a time prior to the point-in-time the PDB was recovered on the primary. See “Performing Recovery When Flashback is Not Enabled” below.

With either method, a PDB that has undergone PDB PITR or flashback on the primary cannot be opened on a standby until it has caught up with the primary.

Performing Recovery When Flashback Is Enabled

If flashback is enabled on the standby, you can flashback the PDB on the standby and then restart standby managed recovery.

1. Determine the affected PDB and PITR SCN.

   The name of the PDB for which recovery was halted is shown in the ORA-39874 message and the PITR SCN is shown in the ORA-39873 message.

2. Close the standby database, if it is still open:

   ```
   SQL> ALTER DATABASE CLOSE;
   ```

3. Flashback the pluggable database on the standby:

   ```
   SQL> FLASHBACK PLUGGABLE DATABASE pdb1 TO SCN 1437260;
   ```

The SCN for the `FLASHBACK PLUGGABLE DATABASE` command is `1437260`, not `1437261` as in the following example, because `TO SCN` and `UNTIL SCN` have different semantics.

4. Restart media recovery on the standby:

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

Performing Recovery When Flashback Is Not Enabled

1. Determine the affected PDB and PITR SCN.

   The name of the PDB for which recovery was halted is shown in the `ORA-39874` message and the PITR SCN is shown in the `ORA-39873` message.

2. Close the standby database, if it is still open:

```
SQL> ALTER DATABASE CLOSE;
```

3. Restore the PDB data files:

```
RMAN> RESTORE PLUGGABLE DATABASE pdb1 UNTIL SCN 1437261;
```

   The `UNTIL SCN` syntax allows RMAN to automatically choose a suitable backup to restore from. After the data files have been restored at the standby, restart MRP to continue applying the redo logs.

4. Restart media recovery on the standby:

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

See Also:

- *Oracle Database Backup and Recovery User's Guide* for more information about performing point-in-time recovery of PDBs

Part II
Reference

The following topics provide reference material to be used in conjunction with the Oracle Data Guard standby database features. For more detailed reference material, refer to the Oracle Database documentation set.

16
Initialization Parameters

This is an alphabetical listing of the initialization parameters you set for a Data Guard environment. It indicates if the parameter is set for the primary role or the standby role, and also provides tips for setting each parameter.

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 16-1 Initialization Parameters for Instances in an Oracle 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 Create a Parameter File for the Standby Database for an example. A logical standby database can have a higher `COMPATIBLE` setting than the primary database if a switchover is not expected. For rolling upgrades using SQL Apply, set this parameter according to the guidelines described in 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 Create a Parameter File for the Standby Database for an example.
`DB_FILE_NAME_CONVERT` = '*location_of_primary_database_datafile*','*location_of_standby_database_datafile*'	Physical Standby Snapshot Standby	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.

Table 16-1 (Cont.) Initialization Parameters for Instances in an Oracle Data Guard Configuration

Parameter	Applicable To	Notes and Recommendations
`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	This parameter is no longer required. If it is not set, the fetch archive log (FAL) server obtains the client's network address from the `LOG_ARCHIVE_DEST_`*n* parameter that corresponds to the client's `DB_UNIQUE_NAME`.
`FAL_SERVER` = *Oracle_Net_service_name*	Physical Standby Snapshot Standby	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 Create a Parameter File for the Standby Database for an example.
`LOG_ARCHIVE_CONFIG` `='DG_CONFIG=(`*db_unique_name, db_unique_name, ...* `)'`	Primary Logical Standby Physical Standby Snapshot Standby	Highly recommended. The `DG_CONFIG` attribute of this parameter must be explicitly set on each database in an Oracle Data Guard configuration to enable full Oracle Data Guard functionality. Set `DG_CONFIG` to a text string that contains the `DB_UNIQUE_NAME` of each database in the configuration, with each name in this list separated by a comma.
`LOG_ARCHIVE_DEST_`*n* = {`LOCATION=`*path_name* | `SERVICE=`*service_name, attribute, attribute, ...*}`	Primary Logical Standby Physical Standby Snapshot Standby	Required. Define up to thirty (where *n* = 1, 2, 3, ... 31) 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 LOG_ARCHIVE_DEST_n Parameter Attributes.
`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.
`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.

Table 16-1 (Cont.) Initialization Parameters for Instances in an Oracle Data Guard Configuration

Parameter	Applicable To	Notes and Recommendations
`LOG_ARCHIVE_MIN_SUCCEED_DEST`	Primary Logical Standby Physical Standby Snapshot Standby	Optional. This parameter specifies the number of local or remote MANDATORY destinations, or local OPTIONAL destinations, that a logfile group must be archived to before it can be re-used.
`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 Setting Archive Tracing.
`LOG_FILE_NAME_CONVERT = '`*location_of_primary_database_redo_logs*`','`*location_of_standby_database_redo_logs*`'`	Logical Standby Physical Standby Snapshot Standby	This parameter converts the path names of the primary database online redo log file to path names on the standby database. See Create a Parameter File for the Standby Database 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 an Oracle 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 data files are added to or dropped from the primary database, corresponding changes are made in the standby database without manual intervention. If the directory structures on the primary and standby databases are different, you must also set the `DB_FILE_NAME_CONVERT` initialization parameter to convert the filenames of one or more sets of data files on the primary database to filenames on the (physical) standby database.

17

LOG_ARCHIVE_DEST_n Parameter Attributes

This is a list of the attributes for the `LOG_ARCHIVE_DEST_n` initialization parameter, (where *n* is an integer between 1 and 31).

- AFFIRM and NOAFFIRM
- ALTERNATE
- COMPRESSION
- DB_UNIQUE_NAME
- DELAY
- ENCRYPTION
- GROUP
- LOCATION and SERVICE (`LOCATION` is not supported for `LOG_ARCHIVE_DEST_11` through `LOG_ARCHIVE_DEST_31`)
- MANDATORY (not supported for `LOG_ARCHIVE_DEST_11` through `LOG_ARCHIVE_DEST_31`)
- MAX_CONNECTIONS
- MAX_FAILURE
- NET_TIMEOUT
- NOREGISTER
- PRIORITY
- REOPEN
- SYNC and ASYNC (`SYNC` is not supported for `LOG_ARCHIVE_DEST_11` through `LOG_ARCHIVE_DEST_31`)
- TEMPLATE
- VALID_FOR

Usage Notes

- Each database in an Oracle Data Guard configuration typically has one destination with the `LOCATION` attribute for the archival of the online and standby redo logs and one destination with the `REMOTE` attribute for every other database in the configuration.
- If configured, each `LOG_ARCHIVE_DEST_1` through `LOG_ARCHIVE_DEST_10` destination must contain either a `LOCATION` or `SERVICE` attribute to specify a local disk directory or a remotely accessed database, respectively. Each `LOG_ARCHIVE_DEST_11` through `LOG_ARCHIVE_DEST_31` destination must contain a `SERVICE` attribute.

  All other attributes are optional.

- `LOG_ARCHIVE_DEST_11` through `LOG_ARCHIVE_DEST_31` can only be used when the `COMPATIBLE` initialization parameter is set to 11.2.0.0 or later.

> **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:**
>
> Redo Transport Services for more information about defining `LOG_ARCHIVE_DEST_n` destinations and setting up redo transport services

17.1 AFFIRM and NOAFFIRM

The `AFFIRM` and `NOAFFIRM` attributes control whether a redo transport destination acknowledges received redo data before or after writing it to the standby redo log.

Definitions of each option are as follows:

- `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, then 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

Example

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
```

17.2 ALTERNATE

The ALTERNATE attribute specifies an alternate archiving destination to be used when the original destination fails.

> **Note:**
>
> As of Oracle Database 12*c* Release 2 (12.2.0.1), you can expand the number of alternate archive destinations and functionality by using the GROUP and PRIORITY attributes in place of the ALTERNATE attribute for remote log archive destinations. This new method cannot be used in conjunction with the ALTERNATE attribute method. For more information, see Alternate Destinations. The ALTERNATE attribute is reserved for configuring alternate local archiving destinations. For backwards compatibility, examples of using ALTERNATE for remote log archiving destination are provided in Using the ALTERNATE Attribute to Configure Remote Alternate Destinations.

Category	ALTERNATE=LOG_ARCHIVE_DEST_*n*
Data Type	String
Valid Value	A LOG_ARCHIVE_DEST_*n* destination, where *n* is a value from 1 through 10.
Default Value	None. If an alternate destination is not specified, then redo transport services do not automatically change to another destination.
Requires attributes	None[1]
Conflicts with attributes	GROUP and PRIORITY[2]
Corresponds to	ALTERNATE and STATUS columns of the V$ARCHIVE_DEST view

1 Although it is not mandatory that MAX_FAILURE be used with ALTERNATE, a nonzero MAX_FAILURE value is required for an alternate to become active. Using ALTERNATE with the default value of MAX_FAILURE (zero), does not result in any change in behavior.

2 If the REOPEN attribute is specified with a nonzero value, then an alternate is not activated until the number of failures is greater than or equal to the value of MAX_FAILURE, with a minimum time period between attempts equal to the value of REOPEN (in seconds).

Usage Notes

- The `ALTERNATE` attribute is optional. If an alternate destination is not specified, then archiving services do not automatically change to another destination if the original destination fails.
- You can specify only one alternate destination for each local `LOG_ARCHIVE_DEST_`*n* parameter (`LOCATION=`…).
- An alternate destination should specify a different disk location on the same local primary or standby database system, as shown in the examples below.
- To configure an alternate destination, set its `LOG_ARCHIVE_DEST_STATE_`*n* parameter to `ALTERNATE`.
- To manually enable an alternate destination, set its `LOG_ARCHIVE_DEST_STATE_`*n* parameter to `ENABLE`.
- If no enabled destinations reference an alternate destination, then the alternate destination is assumed 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 the `ALTER SYSTEM` statement.
- Any destination can be designated as an alternate destination, given the following restrictions:
  - At least one local 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.
- 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.
- If an alternate destination is not specified, or if `MAX_FAILURE` is set to zero (the default), then archiving services do not automatically change to another destination if the original destination fails.

Examples

These examples are included to illustrate basic concepts and are not meant to be used exactly as shown. They will be different in your configuration depending on your local archiving setup.

The following example shows a sample initialization parameter file in which a local archiving destination `LOG_ARCHIVE_DEST_1` automatically fails over to the alternate destination `LOG_ARCHIVE_DEST_2` on the next archival operation if an error occurs, such as a write failure or an allocation failure if the device were to become full.

Example 17-1 Automatically Failing Over to an Alternate Local Destination

```
LOG_ARCHIVE_DEST_1='LOCATION=/disk1 MANDATORY MAX_FAILURE=1
ALTERNATE=LOG_ARCHIVE_DEST_2'

LOG_ARCHIVE_DEST_STATE_1=ENABLE

LOG_ARCHIVE_DEST_STATE_2=ALTERNATE
```

```
LOG_ARCHIVE_DEST_2='LOCATION=/disk2 MANDATORY'
```

To resume archiving to the original destination, LOG_ARCHIVE_DEST_1, you must re-enable it manually. Then you must reset LOG_ARCHIVE_DEST_2 to its former alternate state to avoid having two active local archiving destinations. To do this, set the LOG_ARCHIVE_DEST_STATE_*n* parameters back to their original values, as follows:

```
ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_1=ENABLE

ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2=ALTERNATE
```

You can automate this fallback mechanism. Pair the original destination and the alternate destination by specifying an ALTERNATE attribute that points back to the preferred destination, as shown in the sample initialization parameter file in Example 17-2.

Example 17-2 Automatic Local Alternate Fallback

```
LOG_ARCHIVE_DEST_1='LOCATION=/disk1 MANDATORY MAX_FAILURE=1
ALTERNATE=LOG_ARCHIVE_DEST_2'

LOG_ARCHIVE_DEST_STATE_1=ENABLE

LOG_ARCHIVE_DEST_STATE_2=ALTERNATE

LOG_ARCHIVE_DEST_2='LOCATION=/disk2 MANDATORY
ALTERNATE=LOG_ARCHIVE_DEST_1'
```

If LOG_ARCHIVE_DEST_1 becomes available again, then Oracle Data Guard automatically sets it to become the active local archiving destination and resets LOG_ARCHIVE_DEST_2 as its alternate.

17.3 COMPRESSION

The COMPRESSION attribute is used to specify whether redo data is compressed before transmission to a redo transport destination.

> **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 | DISABLE | ZLIB | LZO]
Data Type	Boolean
Valid values	ENABLE, DISABLE, ZLIB, or LZO
Default value	DISABLE
Requires attributes	None
Conflicts with attributes	None

Category	COMPRESSION=[ENABLE | DISABLE | ZLIB | LZO]
Corresponds to	COMPRESSION column of the V$ARCHIVE_DEST view

Usage Notes

- The ENABLE option enables compression and uses the ZLIB compression algorithm by default.
- The COMPRESSION attribute is optional. If it is not specified, the default compression behavior is DISABLE.
- For Oracle Data Guard SYNC connection strings that also use the Oracle Data Guard COMPRESSION attribute, be sure the SQLNET.COMPRESSION configuration parameter is set to disabled (set to off) in the sqlnet.ora file. See *Oracle Database Net Services Reference* for more information about the SQLNET.COMPRESSION parameter.

Example

The following example shows the COMPRESSION attribute with the LOG_ARCHIVE_DEST_*n* parameter. Since the ENABLE option is specified, the ZLIB compression algorithm is used.

```
LOG_ARCHIVE_DEST_3='SERVICE=denver SYNC COMPRESSION=ENABLE'
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

17.4 DB_UNIQUE_NAME

The DB_UNIQUE_NAME attribute 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 Oracle 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'
```

17.5 DELAY

The DELAY attribute specifies a minimum time lag between when redo data from the primary site is archived on a standby site and when the archived redo log file is applied to the standby database or any standbys cascaded from it.

Category	DELAY[=*minutes*]
Data Type	Numeric
Valid values	>=0 minutes
Default Value	30 minutes
Requires attributes	SERVICE
Conflicts with attributes	LOCATION, VALID_FOR=(*,STANDBY_ROLE)
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, then any delay that you set is 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;
    ```

- The DELAY value that a cascaded standby uses is the value that was set for the LOG_ARCHIVE_DEST_*n* parameter on the primary that shipped the redo to the cascading standby.

> **See Also:**
>
> *Oracle Database SQL Language Reference* for more information about these ALTER DATABASE statements

Example

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=/arch/dest MANDATORY'
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*.

17.6 ENCRYPTION

The `ENCRYPTION` attribute is used to specify whether redo data is encrypted before transmission to a Zero Data Loss Recovery Appliance (Recovery Appliance).

> **Note:**
>
> Redo transport encryption is allowed only for connections to a Recovery Appliance. Attempting to configure encryption on a log archive destination other than a Recovery Appliance results in an error.

Category	ENCRYPTION=ENABLE or DISABLE
Data type	Boolean
Valid values	`ENABLE` or `DISABLE`
Default value	`DISABLE`
Requires attributes	`SERVICE`
Conflicts with attributes	`COMPRESSION`, `SYNC`, `LOCATION`, and `MAX_CONNECTIONS`
Corresponds to	`ENCRYPTION` column of the `V$ARCHIVE_DEST` view

Usage Notes

- The `ENCRYPTION` attribute is optional. If it is not specified, then the default encryption behavior is `DISABLE`.
- To use the `ENCRYPTION` attribute, you must set the `COMPATIBLE` initialization parameter to 11.2.0.4 or higher on the protected database.

Example

The following example shows the `ENCRYPTION` attribute specified on the `LOG_ARCHIVE_DEST_`*n* parameter.

```
LOG_ARCHIVE_DEST_3='SERVICE=denver ENCRYPTION=ENABLE'
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

17.7 GROUP

The `GROUP` attribute is used to specify membership in a specific collection of log archive destinations.

Groups are numbered 1 through 8. The default group (GROUP=0) is special in that it cannot be assigned. The default group is populated with all destinations that are not explicitly assigned to a group.

Category	GROUP=integer
Data Type	Integer
Valid Value	1 through 8
Default Value	0
Requires Attributes	SERVICE
Conflicts with Attributes	ALTERNATE
Corresponds to	Not applicable

Usage Notes

- None

Examples

The following example is given to illustrate basic concepts and is not meant to be used exactly as shown. Depending on your configuration, there may be other parameters, such as DB_UNIQUE_NAME, that are required.

```
LOG_ARCHIVE_DEST_1='SERVICE=FS1 GROUP=1'
LOG_ARCHIVE_DEST_2='SERVICE=FS2 GROUP=1'
LOG_ARCHIVE_DEST_3='SERVICE=FS3 GROUP=2'
LOG_ARCHIVE_DEST_4='SERVICE=FS4 GROUP=2'
```

> **See Also:**
>
> - Assigning Log Archive Destinations to a Group

17.8 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.

LOG_ARCHIVE_DEST_1 through LOG_ARCHIVE_DEST_10 destinations can contain either a LOCATION attribute or a SERVICE attribute.

LOG_ARCHIVE_DEST_11 through LOG_ARCHIVE_DEST_31 destinations can only contain a SERVICE attribute.

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

Category	LOCATION=*local_disk_directory* or USE_DB_RECOVERY_FILE_DEST	SERVICE=*net_service_name*
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.
- The LOG_ARCHIVE_DEST_11 through LOG_ARCHIVE_DEST_31 parameters do not support the LOCATION attribute.
- 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 if media recovery of a database becomes necessary. You can specify up to thirty 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 fast recovery area, specify the directory or Oracle Storage Manager disk group to serve as the fast recovery area using the DB_RECOVERY_FILE_DEST initialization parameter.
- 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 is 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

The following example shows how to specify the LOCATION attribute:

```
LOG_ARCHIVE_DEST_2='LOCATION=/disk1/oracle/oradata/payroll/arch/'
LOG_ARCHIVE_DEST_STATE_2=ENABLE
```

The following example shows how to specify the `SERVICE` attribute:

```
LOG_ARCHIVE_DEST_3='SERVICE=stby1'
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

17.9 MANDATORY

The `MANDATORY` attribute 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

- The `LOG_ARCHIVE_DEST_11` through `LOG_ARCHIVE_DEST_31` parameters do not support the `MANDATORY` attribute.
- 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

Example

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
```

17.10 MAX_CONNECTIONS

The MAX_CONNECTIONS attribute 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 20
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.
- Redo that is received from an ARC*n* process is written directly to an archived redo log file. Therefore, it cannot be applied in real-time as it is received.
- 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 Oracle Data Guard uses 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 returns to the default behavior for remote archival, which is to transport redo data using a single ARC*n* process.

Example

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
```

17.11 MAX_FAILURE

The MAX_FAILURE attribute controls the consecutive number of times at a log switch that redo transport services attempts to reestablish communication and transmit redo data to a failed destination before the primary database gives up on the destination.

The MAX_FAILURE attribute is handled differently in Oracle Database 12*c* Release 1 (12.1) and Oracle Database 12*c* Release 2 (12.2). It is important to understand the differences. See the Usage Notes below.

Category	MAX_FAILURE=*count*
Data type	Numeric
Valid value	>=0
Default value	For default group destinations the default value is 0. For non-default log archive destination group destinations, the default value is 1.
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 for MAX_FAILURE in Oracle Database 12c Release 2 (12.2)

- For redo destinations that use the new `GROUP` and `PRIORITY` attributes, if the error count reaches the value specified for the `MAX_FAILURE` attribute, then the destination enters the ERROR state where it remains until it is found to be accessible. It is checked periodically depending on the value specified for the `REOPEN` attribute.
- For default destinations in log archive groups (those redo destinations that do not use the new `GROUP` and `PRIORITY` attributes), the behavior of the `MAX_FAILURE` attribute is the same as it is in Oracle Database 12*c* Release 1 (12.1.0.1)

Usage Notes for MAX_FAILURE in Oracle Database 12c Release 1 (12.1)

- 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 at log switch 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 set the `LOG_ARCHIVE_DEST_`*n* parameter. 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.
- For log archive destinations not configured as a preferred alternate, if the error count reaches the value specified for the `MAX_FAILURE` attribute, then the destination is disabled and there is no further access until the destination is manually reenabled.
- For log archive destinations configured as a preferred alternate, if the error count reaches the value specified for the `MAX_FAILURE` attribute, then the alternate destination is enabled and the failing destination is switched to the `ALTERNATE` state.

Because this destination is a preferred alternate, it is checked periodically (depending on the value of the REOPEN attribute).

- For log archive destinations configured as a non-preferred alternate, if the error count reaches the value specified for the MAX_FAILURE attribute, then the destination is disabled and there is no further access until the destination is manually (re)enabled. Also, the previously preferred destination (currently unavailable and in the ALTERNATE state) remains in the ALTERNATE state and does not return to service until it is explicitly manually (re)enabled.

Example

The following example allows redo transport services to try reconnecting up to three consecutive times at log switch to the failed destination, as long as each log switch is more than 5 seconds apart. If the archival operation fails after the third attempt, then the destination is treated as if the REOPEN attribute was not specified and the destination is marked as permanently failed until reset.

```
LOG_ARCHIVE_DEST_1='LOCATION=/arc_dest REOPEN=5 MAX_FAILURE=3'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
```

17.12 NET_TIMEOUT

The NET_TIMEOUT attribute specifies the number of seconds that the LGWR background process blocks 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 is 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.
- As of Oracle Database 12*c* Release 12.2 (12.2.0.1), there is a new database initialization parameter, DATA_GUARD_SYNC_LATENCY, which is global for all synchronous standby destinations. It defines the maximum amount of time (in

seconds) that the primary database may wait before disconnecting subsequent destinations after at least one synchronous standby has acknowledged receipt of the redo. See *Oracle Database Reference*.

Example

The following example shows how to specify a 10-second network timeout value on the primary database with the `NET_TIMEOUT` attribute.

```
LOG_ARCHIVE_DEST_2='SERVICE=stby1 SYNC NET_TIMEOUT=10'
LOG_ARCHIVE_DEST_STATE_2=ENABLE
```

17.13 NOREGISTER

The `NOREGISTER` attribute 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 an Oracle Data Guard configuration.
- The `NOREGISTER` attribute *is required* if the destination is not part of an Oracle 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.

Example

The following example shows the `NOREGISTER` attribute:

```
LOG_ARCHIVE_DEST_5='NOREGISTER'
```

17.14 PRIORITY

The `PRIORITY` attribute is used to specify preference within a collection of log archive destinations.

Priorities are numbered 1 through 8. A lower value represents a higher priority. The lowest priority (`PRIORITY=8`) is special in the sense that if that priority is active then all destinations at that priority are made active. If any higher priority destination returns to service, then that destination is made active and all low priority destinations are made inactive.

Category	Priority=*integer*
Data Type	Integer
Valid Value	1 through 8
Default Value	1
Requires attributes	SERVICE
Conflicts with attributes	ALTERNATE
Corresponds to	Not applicable

Usage Notes

- The PRIORITY attribute is always used in conjunction with the GROUP attribute to provide an orderly enabling and fallback of members (redo destinations) of the group.

Example

The following example is given to illustrate basic concepts and is not meant to be used exactly as shown. Depending on your configuration, there may be other parameters, such as DB_UNIQUE_NAME, that are required. A sample log archive destination setup that defines priorities is as follows:

```
LOG_ARCHIVE_DEST_1='SERVICE=FS1 SYNC GROUP=1 PRIORITY=1'
LOG_ARCHIVE_DEST_2='SERVICE=FS2 SYNC GROUP=1 PRIORITY=1'
LOG_ARCHIVE_DEST_3='SERVICE=FS3 ASYNC GROUP=1 PRIORITY=2'
LOG_ARCHIVE_DEST_4='SERVICE=TS ASYNC GROUP=1 PRIORITY=3'
```

This declaration results in the following behavior:

- The primary ships to either of two preferred far sync instances, FS1 or FS2.
- If both FS1 and FS2 become unavailable, then the primary ships to FS3 (in this case via ASYNC).
- If either FS1 or FS2 become available while the primary is shipping to FS3, then the primary fails back to the available preferred log archive destination.
- If all three higher priority log archive destinations fail, the primary begins shipping to TS (Terminal Standby). While shipping to TS, if FS1, FS2, or FS3 become available, then the primary switches to the newly available higher priority destination.

> **See Also:**
>
> - **Assigning Priorities to Log Archive Destinations in a Group**

17.15 REOPEN

The REOPEN attribute specifies the minimum number of seconds before redo transport services 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, then 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, then redo transport services stall the primary database when it is not possible to successfully transmit redo data. When this situation occurs, consider the following options:
  - Change the destination by deferring the destination, specifying the destination as optional, or changing the `SERVICE` attribute value.
  - Specify an alternate destination.
  - Disable the destination.

Example

The following example shows the `REOPEN` attribute.

```
LOG_ARCHIVE_DEST_3='SERVICE=stby1 MANDATORY REOPEN=60'
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

17.16 SYNC and ASYNC

The `SYNC` and `ASYNC` attributes specify 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`

Category	SYNC	ASYNC
Corresponds to	TRANSMIT_MODE column of the V$ARCHIVE_DEST view	TRANSMIT_MODE column of the V$ARCHIVE_DEST view

Usage Notes

- The LOG_ARCHIVE_DEST_11 through LOG_ARCHIVE_DEST_31 parameters do not support the SYNC attribute.
- 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.
- On primary databases and logical standbys, destinations 1 through 10 default to ASYNC (real-time cascading).

  On physical standbys, snapshot standbys, and far sync instances, destinations 1 through 10 default to ARCH transport mode. (Note that the ARCH attribute is deprecated; the use of ARCH in this situation simply indicates non-real-time cascading.)

  Destinations 11 through 31 always default to ASYNC.

> **See Also:**
>
> - *Oracle Database Reference* for more information about LOG_ARCHIVE_DEST_*n* deprecated attributes

Example

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
```

17.17 TEMPLATE

The TEMPLATE attribute defines a directory specification and format template for names of archived redo log files at the destination.

The template is used to generate a filename that is different from the default filename format defined by the LOG_ARCHIVE_FORMAT initialization parameter at the redo destination.

Category	TEMPLATE=filename_template_%t_%s_%r
Data Type	String value
Valid values	Not applicable
Default value	None
Requires attributes ...	SERVICE
Conflicts with attributes ...	LOCATION

Category	TEMPLATE=filename_template_%t_%s_%r
Corresponds to ...	REMOTE_TEMPLATE and REGISTER columns of the V$ARCHIVE_DEST view

Usage Notes

- The TEMPLATE attribute is optional. If TEMPLATE is not specified, archived redo logs are named using the value of the LOG_ARCHIVE_FORMAT initialization parameter.
- The TEMPLATE attribute overrides the LOG_ARCHIVE_FORMAT initialization parameter setting at the remote archival destination.
- The TEMPLATE attribute is valid only with remote destinations (specified with the SERVICE attribute).
- The value you specify for *filename_template* must contain the %s, %t, and %r directives described in Table 17-1.

Table 17-1 Directives for the TEMPLATE Attribute

Directive	Description
%t	Substitute the instance thread number.
%T	Substitute the instance thread number, zero filled.
%s	Substitute the log file sequence number.
%S	Substitute the log file sequence number, zero filled.
%r	Substitute the resetlogs ID.
%R	Substitute the resetlogs ID, zero filled.

- The filename_template value is transmitted to the destination, where it is translated and validated before creating the filename.

17.18 VALID_FOR

The VALID_FOR attribute specifies whether redo data gets written to a destination.

The following factors are considered:

- 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 an Oracle 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.

18

SQL Statements Relevant to Oracle Data Guard

There are many SQL and SQL*Plus statements that are useful for performing operations on standby databases in an Oracle Data Guard environment.

See the following topics:

- ALTER DATABASE Statements
- ALTER SESSION Statements
- ALTER SYSTEM Statements

Refer to the *Oracle Database SQL Language Reference* for complete syntax and descriptions of all SQL statements.

See Initialization Parameters for a list of initialization parameters that you can set and dynamically update using the `ALTER SYSTEM SET` statement.

18.1 ALTER DATABASE Statements

This table describes `ALTER DATABASE` statements that are relevant to Oracle Data Guard.

Table 18-1 ALTER DATABASE Statements Used in Data Guard Environments

ALTER DATABASE Statement	Description
`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. • For logical standby databases, use the `ALTER DATABASE PREPARE TO SWITCHOVER` and `ALTER DATABASE COMMIT TO SWITCHOVER` statements.
`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 Add or Drop a Redo Log File Group 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.

Table 18-1 (Cont.) ALTER DATABASE Statements Used in Data Guard Environments

ALTER DATABASE Statement	Description
`[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.
`COMMIT TO SWITCHOVER`	Performs a switchover to: • Change the current primary database to the standby database role • Change one standby database to the primary database role. When switching over to a physical standby database, as of Oracle Database 12*c* Release 1 (12.1), the `COMMIT TO SWITCHOVER` statement has been replaced with the `SWITCHOVER TO` statement. The `COMMIT TO SWITCHOVER` statement is still supported, but Oracle recommends that you use the new `SWITCHOVER TO` statement. **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 Performing a Switchover to a Physical Standby Database Using Old Syntax and Performing a Failover to a Physical Standby Database Using Old Syntax 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 Create a Control File for the Standby Database 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 Add or Drop a Redo Log File Group 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.
`FAILOVER TO` *target_db_name*	This statement is for physical standby databases only. It initiates a failover to the specified host database.
`[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 at least be mounted (and it can also be open) when you issue this statement. See Enable Forced Logging for an example of this statement.

Table 18-1 (Cont.) ALTER DATABASE Statements Used in Data Guard Environments

ALTER DATABASE Statement	Description
`GUARD`	Controls user access to tables in a logical standby database. Possible values are `ALL`, `STANDBY`, and `NONE`. See Controlling User Access to Tables in a Logical Standby Database 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`	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 Performing a Switchover to a Logical Standby Database for examples of this statement.
`RECOVER MANAGED STANDBY DATABASE [ { DISCONNECT [FROM SESSION] | PARALLEL n| 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 3 in Start the Physical Standby Database and Applying Redo Data to Physical Standby Databases 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 deprecated 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. **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.

Table 18-1 (Cont.) ALTER DATABASE Statements Used in Data Guard Environments

ALTER DATABASE Statement	Description
`REGISTER [OR REPLACE] [PHYSICAL|LOGICAL] LOGFILE` *`filespec`*	Allows the registration of manually copied archived redo log files. **Note**: Issue this command only after manually copying the corresponding archived redo log file to the standby database. Issuing this command while the log file is in the process of being copied or when the log file does not exist may result in errors on the standby database at a later time.
`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 Convert to a Logical Standby Database 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 Oracle Data Guard configuration. You specify this clause from the primary database.
`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 Starting SQL Apply 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 Performing a Failover to a Logical Standby Database for an example of this statement.
`SWITCHOVER TO` *`target_db_name`*	This statement is for physical standby databases only. It initiates a switchover on the primary database to the specified physical standby database.

18.2 ALTER SESSION Statements

This table describes the `ALTER SESSION` statements that are relevant to Oracle Data Guard.

Table 18-2 ALTER SESSION Statements Used in Oracle 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 Modifying a Logical Standby Database for more information.

Table 18-2 (Cont.) ALTER SESSION Statements Used in Oracle Data Guard Environments

ALTER SESSION Statement	Description
`ALTER SESSION SYNC WITH PRIMARY`	This statement is for physical standby databases only. This statement synchronizes a physical standby database with the primary database, by blocking until all redo data received by the physical standby at the time of statement invocation has been applied. See Forcing Redo Apply Synchronization in a Real-time Query Environment for more information.

18.3 ALTER SYSTEM Statements

This table describes the `ALTER SYSTEM` statements that are relevant to Oracle Data Guard.

Table 18-3 ALTER SYSTEM Statements Used in Oracle Data Guard Environments

ALTER SYSTEM Statement	Description
`ALTER SYSTEM FLUSH REDO TO target_db_name [[NO] CONFIRM APPLY]`	This statement flushes redo data from a primary database to a standby database and optionally waits for the flushed redo data to be applied to a physical or logical standby database. This statement must be issued on a mounted, but not open, primary database.

19
Views Relevant to Oracle Data Guard

There are a number of views that are especially useful when monitoring an Oracle Data Guard environment.

Table 19-1 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 19-1 Views That Are Pertinent to Oracle 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 an Oracle 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 to 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.
DBA_ROLLING_UNSUPPORTED	Logical only	Displays the schemas, tables, and columns in those tables, that contain unsupported data types for a rolling upgrade operation for a logical standby database using the DBMS_ROLLING PL/SQL package. Use this view before you perform a rolling upgrade using DBMS_ROLLING to determine what is unsupported.
V$ARCHIVE_DEST	Primary, physical, snapshot, and logical	Describes all of the destinations in the Oracle Data Guard configuration, including each destination's current value, mode, and status. **Note:** The information in this view does not persist across an instance shutdown.
V$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 19-1 (Cont.) Views That Are Pertinent to Oracle 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$DATABASE	Primary, physical, snapshot, and logical	Provides database information from the control file. Includes information about fast-start failover (available only with the Oracle Data Guard broker).
V$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 data file 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 various Oracle Data Guard statistics, including apply lag and transport lag. This view can be queried on any instance of a standby database. No rows are returned if queried on a primary database. See also Choosing a Target Standby Database for a Role Transition 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$LOG_HISTORY	Primary, physical, snapshot, and logical	Contains log history information from the control file.
V$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.

Table 19-1 (Cont.) Views That Are Pertinent to Oracle Data Guard Configurations

View	Database	Description
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. **Note:** The information in this view does not persist across an instance shutdown.
V$STANDBY_EVENT_HISTOGRAM	Physical	Contains a histogram of apply lag values for the physical standby. An entry is made in the corresponding apply lag bucket by the Redo Apply process every second. (This view returns rows only on a physical standby database that has been open in real-time query mode.) **Note:** The information in this view does not persist across an instance shutdown.
V$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:

A
Troubleshooting Oracle Data Guard

These are some of the problems that can occur on a standby database, and the troubleshooting procedures to address them.

A.1 Common Problems

These are some of the common problems you may encounter when using a standby database.

A.1.1 Renaming Data Files with the ALTER DATABASE Statement

You cannot rename the data file 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 Adding a Data File or Creating a Tablespace to learn how to add data files 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", -
> STATUS "DB_status", -
> DESTINATION "Archive_dest", -
> ERROR "Error" -
> 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, then check the following list of possible issues. If any of the following conditions exist, then redo transport services 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 automatically fails 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 automatically switches 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

These are some of the problems that can cause a switchover to be unsuccessful, and possible solutions.

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 is applied automatically. Query the `SWITCHOVER_STATUS` column in the `V$DATABASE` view. A switchover to the primary role is now possible if the `SWITCHOVER_STATUS` column returns `TO PRIMARY` or `SESSIONS ACTIVE`.

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

SWITCHOVER_STATUS
-----------------
TO PRIMARY
1 row selected
```

To continue with the switchover, follow the instructions in Performing a Switchover to a Physical Standby Database for physical standby databases or Performing a Switchover to a Logical Standby Database for logical standby databases, and try again to switch the target standby database to the primary role.

A.4.2 Switchover Fails with the ORA-01102 Error

This is an explanation of why you may receive an ORA-01102 error during a switchover, and what action to take.

Suppose the standby database and the primary database reside on the same site. After the ALTER DATABASE SWITCHOVER TO *target_db_name* statement is 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 (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.4.3 Redo Data Is Not Applied After Switchover

If the archived redo log files are not applied to the new standby database after the switchover, it could be 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 to enable the standby site to automatically add new data files that are created at the primary site.

A.4.4 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.

Take the following steps. (This functionality is available starting with Oracle Database 11*g* Release 2 (11.2.0.2).)

1. Shut down and mount the new standby database (old primary).
2. Start Redo Apply on the new standby database.
3. Verify that the new standby database is ready to be switched back to the primary role. Query the SWITCHOVER_STATUS column of the V$DATABASE view on the new 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 new standby database is ready to be switched to the primary role. Continue to query this column until the value returned is either TO PRIMARY or SESSIONS ACTIVE.

4. Issue the following statement to convert the new standby database back to the primary role:

```
SQL> ALTER DATABASE SWITCHOVER TO target_db_name [FORCE];
```

   If this statement is successful, then the database runs in the primary database role, and you do not need to perform any more steps.

   If this statement is unsuccessful, then continue with Step 5.

5. 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 reverts the new standby database back to the primary role.
6. Shut down the original physical standby database.
7. 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. Create a Control File for the Standby Database describes how to create a physical standby control file.
8. Restart the original physical standby instance.

If this procedure is successful and archive gap management is enabled, then the FAL processes 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 Manual Gap Resolution for information about archive gap management and Setting Archive Tracing for information about locating the trace files.

9. Try the switchover again.

   At this point, the Oracle 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 Oracle 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 Oracle Data Guard broker, then go directly to Failures During the Commit Phase of a Switchover Operation.

> **Note:**
>
> Oracle recommends that Flashback Database be enabled for all databases in an Oracle Data Guard configuration. The steps in this section assume that you have Flashback Database enabled on all databases in your Oracle 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, then 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.

To do so, issue 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 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, there are corrective steps you can take.

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 -
   > 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 Failure While Preparing the Logical Standby Database.
   - 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 Failure While Preparing the Primary Database .

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, there are corrective steps you can take.

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 -
   > 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 Failure While Preparing the Logical Standby Database.
- If the query returns a 1, then the logical standby is in an inconsistent state. 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

When an unsupported statement or package is encountered, SQL Apply stops.

Apply services cannot apply unsupported DML statements, DDL statements, and Oracle supplied packages to a logical standby database running SQL Apply.

If SQL Apply has stopped because of an unsupported statement or package, you can take the actions described in Table A-1 to correct the situation and start SQL Apply on the logical standby database again.

Table A-1 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. It shows 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 Views Relevant to Oracle Data Guard 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 its maximum value of 65535 bytes 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* for information about other ways to change the Oracle NET SDU parameter.

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 shuts down if it is operating in maximum protection mode,

  or

- The RFS process on the standby database creates an archived redo log file on the standby database and writes 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 of adverse effects on the primary database because a standby redo log file of the required size is not available at log switch time.

A.10 Troubleshooting a Logical Standby Database

These troubleshooting tips can help you recover from errors.

- 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, then the DDL executes 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 -
   > 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, 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 data file 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 is 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 because it will skip not only the DML seen in the events table, but all the DML associated with the transaction as well.

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 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 topic 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, you issue the SQL*Plus TRUNCATE TABLE command against the table on the primary database. This has 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 now attempts to DELETE zero rows from the object on the primary database. Because no rows were deleted from the primary database, there is no redo recorded in the redo log files. Therefore, no DELETE statement is 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 issues 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 issues 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 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 -
>  , SS.OWNER -
>  , SS.OBJECT_NAME -
>  , SS.STATISTIC_NAME -
>  , SS.VALUE -
>  FROM V$SEGMENT_STATISTICS SS -
>  , V$LOCK L -
>  , V$STREAMS_APPLY_SERVER SAS -
>  WHERE SAS.SERVER_ID = &SLAVE_ID -
>  AND L.SID = SAS.SID -
>  AND L.TYPE = 'TM' -
>  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) -
>  , DISK_READS -
>  , EXECUTIONS -
>  , DISK_READS/EXECUTIONS -
>  , HASH_VALUE -
>  , ADDRESS -
```

```
>  FROM V$SQLAREA -
>  WHERE DISK_READS/GREATEST(EXECUTIONS,1) > 1 -
>  AND ROWNUM < 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, define an index 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. The following shows an example of the warning messages.

```
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 the above output 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 -
>  FROM V$SQLAREA SA -
  >  , V$SESSION S -
  >  , V$STREAMS_APPLY_SERVER SAS -
  >  WHERE SAS.SERVER_ID = &SLAVEID -
  >  AND S.SID = SAS.SID -
  >  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 -
> FROM V$LOCK -
> 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 is not reported in the standby databases alert log. Therefore, to determine which objects have been subjected to ITL pressure, issue the following statement:

```
SQL> SELECT OWNER, OBJECT_NAME, OBJECT_TYPE -
>  FROM V$SEGMENT_STATISTICS -
>  WHERE STATISTIC_NAME = 'ITL waits' -
>  AND VALUE > 0 -
>  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 Oracle 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 is the initial number of concurrent transaction entries allocated within each data block allocated to the database object

The following example shows the necessary steps to increase the `INITRANS` for table `load_tab_1` in the schema `app`.

1. Stop SQL Apply:

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

2. Temporarily bypass the database guard:

```
SQL> ALTER SESSION DISABLE GUARD;
Session altered.
```

3. Increase the INITRANS on the standby database. For example:

```
SQL> ALTER TABLE APP.LOAD_TAB_1 INITRANS 30;
Table altered
```

4. Reenable the database guard:

```
SQL> ALTER SESSION ENABLE GUARD;
Session altered
```

5. Start SQL Apply:

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

Also, consider modifying the database object on the primary database, so that in the event of a switchover, the error does 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 strategy relies upon the UNDO_RETENTION initialization parameter specified on the standby database instance.

Under normal circumstances, the ORA-1403 error is not 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, you can see 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, query the database 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
```

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, restores 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 is applied to the standby database:

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

B

Patching, Upgrading, and Downgrading Databases in an Oracle Data Guard Configuration

These topics address how to upgrade and downgrade an Oracle database when a physical or logical standby database is present in the Oracle Data Guard configuration.

B.1 Before You Patch or Upgrade the Oracle Database Software

There are several points to consider before beginning to patch or upgrade your Oracle Database software.

- If you are using the Oracle 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 described in these topics are to be used in conjunction with the ones contained in the *Oracle Database Upgrade Guide*.
- Check for nologging operations. If nologging operations have been performed then you must update the standby database. See Recovering After the NOLOGGING Clause Is Specified for details.
- Make note of any tablespaces or data files that need recovery due to `OFFLINE IMMEDIATE`. Tablespaces or data files should be recovered and either online or offline prior to upgrading.
- In an Oracle Data Guard configuration, all physical and snapshot standby databases must use a copy of the password file from the primary database. As of Oracle Database 12*c* Release 2 (12.2.0.1) , password file changes done on the primary database are automatically propagated to standby databases. (Password file changes include when an administrative privilege (`SYSDG`, `SYSOPER`, `SYSDBA`, and so on) is granted or revoked, and when the password of any user with administrative privileges is changed.)

  Far sync instances are an exception to the automatic updating feature. Updated password files must still be manually copied to far sync instances because far

sync instances receive redo, but do not apply it. When a password file is manually updated at a far sync instance, the redo containing the same password changes from the primary database is automatically propagated to any standby databases that are set up to receive redo from that far sync instance. The password file is updated on the standby when the redo is applied.

> **Note:**
>
> If there are cascaded standbys in your configuration then those cascaded standbys must follow the same rules as any other standby but should be shut down last and restarted in the new home first.

B.2 Patching Oracle Database with Standby First Patching

Oracle Data Guard Standby-First Patch Apply provides support for different database home software between a primary database and its physical standby database(s).

This support is provided for the purpose of applying and validating Oracle patches and patch bundles in rolling fashion with minimal risk to the primary database. For example, with Data Guard Standby-First Patch Apply you apply a database home patch first to a physical standby database. The standby is used to run read-only workload, or read-write workload if it is a snapshot standby, for testing and evaluation of the patch. After passing evaluation, the patch is then installed on the primary system with greater assurance of the effectiveness and stability of the database home patch.

Oracle Data Guard Standby-First Patch Apply is supported only for certified interim patches and patch bundles (for example, Patch Set Update, or Database Patch for Exadata) for Oracle Database 11.2.0.1 and later, on both Oracle Engineered Systems (e.g. Exadata, SuperCluster) and non-Engineered Systems. A patch and patch bundle that is Data Guard Standby-First certified states the following in the patch README:

```
Data Guard Standby-First Installable
```

The following types of patches are candidates to be Data Guard Standby-First certified:

- Database home interim patches
- Exadata bundle patches (e.g. Monthly and quarterly database patches for Exadata)
- Database patch set updates

Patches and patch bundles that update modules that may potentially disrupt the interoperability between primary and physical standby systems running different database home software are not certified “Data Guard Standby-First Installable” and do not state so in the patch README.

Oracle patch sets and major release upgrades do not qualify for Data Guard Standby-First Patch Apply. For example, upgrades from 11.2.0.2 to 11.2.0.3 or 11.2 to 12.1 do not qualify. Use the Data Guard transient logical standby rolling upgrade process for database patch sets and major releases.

Additionally, as of Oracle Database 11*g* Release 2 (11.2.0.1), a physical standby database can be used to install eligible one-off patches, patch set updates (PSUs), and critical patch updates (CPUs), in rolling fashion. For more information about this functionality, see the My Oracle Support note 1265700.1 at `http://support.oracle.com`.

B.3 Upgrading Oracle Database with a Physical Standby Database in Place

These steps show how to upgrade to Oracle Database 12*c* Release 2 (12.2) 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. Install the new release of the Oracle software into a new Oracle home on the physical standby database and primary database systems, as described in the *Oracle Database Upgrade Guide*.
3. Shut down the primary database.
4. Shut down the physical standby database(s).
5. Stop all listeners, agents, and other processes running in the Oracle homes that are to be upgraded. Perform this step on all nodes in an Oracle Real Application Clusters (Oracle RAC) environment.
6. If Oracle Automatic Storage Management (Oracle ASM) is in use, shut down all databases that use Oracle ASM, and then shut down all Oracle ASM instance(s).
7. In the new Oracle home, restart all listeners, agents, and other processes that were stopped in step 5.
8. Mount the physical standby database(s) on the new Oracle home (upgraded version). See Start the Physical Standby Database for information on how to start a physical standby database.

> **Note:**
>
> Do not open the standby database(s) until the primary database upgrade is completed.

9. Start Redo Apply on the physical standby database(s). See Start the Physical Standby Database for information on how to start Redo Apply.
10. Upgrade the primary database as described in the *Oracle Database Upgrade Guide*. The physical standby database(s) is upgraded when the redo generated by the primary database as it is upgraded is applied.
11. Open the upgraded primary database.
12. If Oracle Active Data Guard was being used prior to the upgrade, then refer to Real-time query for information about how to reenable it after upgrading.
13. Optionally, modify the `COMPATIBLE` initialization parameter, following the procedure described in Modifying the COMPATIBLE Initialization Parameter After Upgrading.

> **Note:**
>
> On Windows platforms, it is necessary to use the ORADIM utility to delete the database service (for the old database version) and to create a new database service for the new database version. The `OracleService<SID>` must be replaced on both the primary and standby servers.

B.4 Upgrading Oracle Database with a Logical Standby Database in Place

These steps describe the traditional method for upgrading your Oracle Database software with a logical standby database in place.

> **Note:**
>
> Using SQL Apply to Upgrade the Oracle Database presents an alternative method to perform an 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.

Perform the following steps to upgrade to Oracle Database 12*c* Release 2 (12.2) when a logical standby database is present in the configuration. These steps assume that the primary database is running in `MAXIMUM PERFORMANCE` data protection mode.

Review and perform the steps listed in the "Preparing to Upgrade" chapter of the *Oracle Database Upgrade Guide*

1. Set the data protection mode to `MAXIMUM PERFORMANCE` at the primary database, if needed:

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

2. 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;
   ```

3. Stop SQL Apply on the logical standby database:

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

4. On the primary database install the newer release of the Oracle software. .
5. On the logical standby database, install the newer release of the Oracle software.

> **Note:**
>
> Steps 4 and 5 can be performed concurrently (in other words, the primary and the standby databases can be upgraded concurrently) to reduce downtime during the upgrade procedure.

6. 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;
   ```

7. 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;
   ```

8. Optionally, reset to the original data protection mode if you changed it in Step 1.
9. Optionally, modify the `COMPATIBLE` initialization parameter.

Related Topics:

- *Oracle Database Upgrade Guide*

B.5 Modifying the COMPATIBLE Initialization Parameter After Upgrading

When you upgrade to a new release of Oracle Database, certain new features might make your database incompatible with your previous release.

Oracle Database enables you to control the compatibility of your database with the `COMPATIBLE` initialization parameter.

After the upgrade is complete, you can increase the setting of the `COMPATIBLE` initialization parameter to the maximum level for the new Oracle Database release. When you are certain that you no longer need the ability to downgrade your database back to its original version, set the `COMPATIBLE` initialization parameter based on the compatibility level you want for your new database.

In an Oracle Data Guard configuration, if you decide to increase the setting of the `COMPATIBLE` initialization parameter after upgrading, then it is important that you perform the following steps in the order shown (be sure the standby database has a `COMPATIBLE` setting equal to, or higher than, the primary):

1. Increase the value of the `COMPATIBLE` initialization parameter on all standby databases in the configuration first, as follows:
   a. Ensure that apply is current on the standby database(s).
   b. On one instance of each standby database, execute the following SQL statement:

      ```
      ALTER SYSTEM SET COMPATIBLE=<value> SCOPE=SPFILE;
      ```

   c. If Redo Apply or SQL Apply is running, then stop them.
   d. Restart all instances of the standby database(s).

e. If you previously stopped Redo Apply or SQL Apply, then restart them.

2. Increase the value of the `COMPATIBLE` initialization parameter on the primary database, as follows:

   a. On one instance of the primary database, execute the following SQL statement:

      ```
      ALTER SYSTEM SET COMPATIBLE=<value> SCOPE=SPFILE;
      ```

   b. Restart all instances of the primary database.

> **See Also:**
>
> - *Oracle Database Upgrade Guide* for more information about compatibility settings

B.6 Downgrading Oracle Database with No Logical Standby in Place

Perform these steps to downgrade Oracle Database in an Oracle Data Guard configuration that does *not* contain a logical standby database.

1. Ensure that all physical standby databases are mounted, but not open. Do not open the standby database(s) until all redo generated by the downgrade of the primary database has been applied.
2. Start Redo Apply, in real-time apply mode, on the physical standby database(s).
3. Downgrade the primary database using the procedure described in *Oracle Database Upgrade Guide*, keeping the following in mind:
   - At each step of the downgrade procedure where a script is executed, execute the script only at the primary database. Do not perform the next downgrade step until all redo generated by the execution of the script at the primary database has been applied to each physical standby database.
   - At each step of the downgrade procedure where an action other than running a script is performed, perform the step at the primary database first and then at each physical standby database. Do not perform the next downgrade step at the primary database until the action has been performed at each physical standby database.
4. If it becomes necessary to perform a failover during a downgrade, perform the failover and then continue with the downgrade procedure at the new primary database.

B.7 Downgrading Oracle Database with a Logical Standby in Place

Perform these steps to downgrade Oracle Database in an Oracle Data Guard configuration that contains a logical standby database or a mixture of logical and physical standby databases.

1. Issue the following command at the primary database (database P, for the sake of this discussion) before you downgrade it:

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

   Database P is no longer in the primary database role.

2. Wait for *all* standby databases in the configuration to finish applying all available redo. To determine whether each standby database has finished applying all available redo, run the following query at each standby database:

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

   SWITCHOVER_STATUS
   -----------------
   TO PRIMARY
   ```

   Do not continue on to step 3 until the query returns a value of `TO PRIMARY` for all standby databases in the configuration.

3. Downgrade the logical standby databases using the procedures described in *Oracle Database Upgrade Guide*, keeping the following in mind:

   - At each step of the downgrade procedure where a script is executed, execute the script only at the logical standby databases. Do not perform the next downgrade step until all redo generated by executing the script at the logical standby database that was most recently in the primary role (database P) has been applied to each physical standby database.
   - At each step of the downgrade procedure where an action other than running a script is performed, first perform the step at the logical standby database that was most recently in the primary role (database P), and then perform the step at each physical standby database. Do not perform the next downgrade step at the logical standby database that was most recently in the primary role (database P) until the action has been performed at each physical standby database.

4. After the logical standby that was most recently in the primary role (database P) has been successfully downgraded, open it, and issue the following command:

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

   Database P is now back in the primary role.

5. At each of the logical standby databases in the configuration, issue the following command (note that the command requires that a database link back to the primary exist in all of the logical standby databases):

   ```
   SQL> ALTER DATABASE START LOGICAL STANDBY APPLY IMMEDIATE NEW PRIMARY
   prim_db_link;
   ```

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.

The following topics describe the various database objects, storage types, and PL/SQL supplied packages that are supported and unsupported by logical standby databases:

C.1 Datatype Considerations

See these topics for information about supported and unsupported database objects.

C.1.1 Supported Datatypes in a Logical Standby Database

These are the datatypes that logical standby databases support.

Logical standby databases support the following datatypes:

- Abstract Data Types (ADTs) and ADT tables
  - ADTs cannot contain any data types that are not supported as a top-level column type (for example, nested tables, PKREFs, BFILE, unsupported opaque types).
  - For a table with ADT columns to be supported there must be a primary key (or at least a unique constraint or unique index) that consists solely of scalar top-level columns (scalar ADT attributes cannot be part of such a candidate key).
- `BINARY_DOUBLE`
- `BINARY_FLOAT`
- `BLOB`, `CLOB`, and `NCLOB` stored as BasicFile and SecureFiles. SecureFiles can be compressed, encrypted, or deduplicated. SecureFiles support requires that the primary database be running at a compatibility of 11.2 or higher. See Support for SecureFiles LOBs
- `CHAR`
- `DATE`
- `INTERVAL YEAR TO MONTH`
- `INTERVAL DAY TO SECOND`
- `LONG`
- `LONG RAW`
- `NCHAR`
- `NUMBER`
- `NVARCHAR2`
- Objects stored as `VARRAY`s (except for Collections)
- Oracle Text
- `RAW`
- Multimedia (See exceptions listed in Unsupported Datatypes in a Logical Standby Database.):
  - `ORDAudio`
  - `ORDDataSource` (internal)
  - `ORDDicom`
  - `ORDDoc`
  - `ORDImage`
  - `ORDSource` (internal)
  - `ORDVideo`

- Spatial (See exceptions listed in Unsupported Datatypes in a Logical Standby Database.)
- `TIMESTAMP`
- `TIMESTAMP WITH TIMEZONE`
- `TIMESTAMP WITH LOCAL TIMEZONE`
- `VARCHAR` and `VARCHAR2`
- `XMLType` data for all storage models, assuming the following primary database compatibility requirements:
  - `XMLType` stored in `CLOB` format requires that the primary database be run at a compatibility setting of 11.0 or higher (`XMLType` stored as `CLOB` is deprecated as of Oracle Database 12*c* Release 1 (12.1).)
  - `XMLType` stored in object-relational format or as binary XML requires that the primary database be running Oracle Database 11*g* Release 2 (11.2.0.3) or higher with a redo compatibility setting of 11.2.0.3 or higher

> **Note:**
>
> SQL Apply does not support statements that have function calls that perform DML on ADT, `LOB`, or `XMLType` columns.

> **Note:**
>
> As of Oracle Database 12*c* Release 1 (12.1), the maximum size of the `VARCHAR2`, `NVARCHAR2`, and `RAW` datatypes has been increased to 32 KB when the `COMPATIBLE` initialization parameter is set to 12.0 or later and the `MAX_STRING_SIZE` initialization parameter is set to `EXTENDED`. Logical standby databases support this increased size in most cases. See "Ensure Table Rows in the Primary Database Can Be Uniquely Identified" for known restrictions.

C.1.1.1 Compatibility Requirements

SQL Apply support for theses features 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 `LOB`s 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.
- TDE support requires primary database to run at a compatibility of 11.1 or higher.
- Basic compression and advanced row compression require the primary database to run at a compatibility of 11.1 or higher.

- Hybrid Columnar Compression support is dependent on the underlying storage system.

> **See Also:**
>
> - *Oracle Database Concepts* for more information about Hybrid Columnar Compression

C.1.1.2 Opaque Type Restrictions

These are the restrictions regarding opaque types.

- `SYS.ANYDATA` is supported as long as the instance does not store user-defined opaque data types or `BFILES`.
- `SYS.ANYDATASET`, `SYS.ANYTYPE`, and user-defined opaque types are not supported.

C.1.2 Unsupported Datatypes in a Logical Standby Database

Some data types are not supported by logical standby databases.

If a table contains columns having any of the following unsupported data types, then the entire table is ignored by SQL Apply. (See Support for Data Types That Lack Native Redo-Based Support for information about support for data types that lack native redo-based support.)

- `BFILE`
- `ROWID`, `UROWID`
- Nested tables
- Objects with nested tables
- Identity columns

C.2 Support for Data Types That Lack Native Redo-Based Support

The Extended Datatype Support (EDS) feature provides a mechanism for logical standbys to support certain data types that lack native redo-based support.

For example, tables with `SDO_GEOMETRY` columns can be replicated using EDS. (Source tables must have a primary key.)

You can query the `DBA_LOGSTDBY_EDS_SUPPORTED` view to find out which tables are candidates for EDS.

> **See Also:**
>
> - Using Extended Datatype Support During Replication for more information about EDS

C.3 Support for Transparent Data Encryption (TDE)

Oracle 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 replicated on 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 supported for logical standby databases in Oracle Database 11*g* Release 2 (11.2) and later.

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.4 Support for Tablespace Encryption

Oracle 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 Support for Transparent Data Encryption (TDE) apply.

Encryption, re-keying, or decryption of a tablespace on a primary does not trigger the need for the same operation on a logical standby. However, a logical standby must have the capability of re-keying as well.

> **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, then 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.5 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 is transparently 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 does not have its data maintained by Logical Standby, so `DBMS_RLS` and `DBMS_FGA` calls referencing that table are not maintained.

C.5.1 Row-level Security

Row-Level Security, also known as Virtual Private Database (VPD), is a feature that enforces security at a fine level of granularity, when accessing 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.5.2 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.5.3 Skipping and Enabling PL/SQL Replication

PL/SQL can be configured with skip and skip_error rules exactly as DDL statements except that the use of wildcards 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);
```

The schema specified is the schema in which the package is defined. To skip an individual procedure in a package, the syntax is as follows:

```
DBMS_LOGSTDBY.Skip (
stmt => 'PL/SQL',
schema_name => 'SYS',
object_name =>'DBMS_RLS.ADD_POLICY',
use_like => FALSE);
```

To skip VPD on certain schemas or tables, a skip procedure must be used. The skip procedure is 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.6 Oracle Label Security

As of Oracle Database 12*c* Release 2 (12.2), you can upgrade databases that use Oracle Label Security (OLS) to new Oracle Database releases and patch sets using

Oracle Data Guard database rolling upgrades with a transient logical standby database and the PL/SQL package, `DBMS_ROLLING`.

C.7 Oracle Database Vault

Oracle Data Guard rolling upgrades support databases that use Oracle Database Vault.

As of Oracle Database 12*c* Release 2 (12.2.0.1), you can upgrade databases that use Oracle Database Vault to new Oracle Database releases and patch sets by using Oracle Data Guard database rolling upgrades with a transient logical standby and the PL/SQL package, `DBMS_ROLLING`.

C.8 Oracle E-Business Suite

Logical standby databases do not fully support an Oracle E-Business Suite implementation because there are tables that contain unsupported data types.

However, using `SKIP` rules, it is possible for you to replicate a subset of the E-Business Suite schemas and tables to offload applications to the logical standby.

> **See Also:**
>
> The My Oracle Support note 851603.1 at `http://support.oracle.com` for additional information about using Logical standby with Oracle E-Business Suite

C.9 Supported Table Storage Types

Logical standby databases support several 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).
- Advanced row compression and basic table compression. Both of these options require that the compatibility setting of the primary database be set to 11.1.0 or higher.
- Tables containing LOB columns stored as SecureFiles, when compatibility is set to 11.2 or higher.
- Tables using Hybrid Columnar Compression, when compatibility is set to 11.2.0.2 or higher.

> **See Also:**
>
> – *Oracle Database Concepts* for more information about Hybrid Columnar Compression

- Tables with virtual columns (provided the table has no other columns or properties not supported by logical standby)
- If there is no primary key and no non-null unique constraint or index, then all columns with a declared maximum length of 4000 bytes are logged as part of the `UPDATE` statement to help identify the modified row. For the purpose of row identification, logical standby requires that a table have at least one visible (not virtual) column of one of the following datatypes:
  - `CHAR`
  - `VARCHAR`
  - `VARCHAR2` (with a declared column length <= 4000 bytes)
  - `NVARCHAR`
  - `NVARCHAR2` (with a declared column length <= 4000 bytes)
  - `NUMBER`
  - `DATE`
  - `RAW`
  - `BINARY FLOAT`
  - `BINARY DOUBLE`
  - `TIMESTAMP`
  - `TIMESTAMP WITH TIME ZONE`
  - `TIMESTAMP WITH LOCAL TIME ZONE`
  - `INTERVAL YEAR TO MONTH`
  - `INTERVAL DAY TO SECOND`

> **See Also:**
>
> – Ensure Table Rows in the Primary Database Can Be Uniquely Identified

C.10 Unsupported Table Storage Types

If a table contains *only* these datatypes, then logical standby does not support it.

- `LOB` (`CLOB`, `NCLOB`, `BLOB`)
- `LONG`
- `LONG RAW`
- `OBJECT TYPE`

- COLLECTIONS
- XML
- VARCHAR2 (with a declared column length > 4000 bytes)
- NVARCHAR2 (with a declared column length > 4000 bytes)
- RAW (with a declared column length > 4000 bytes)

> **See Also:**
>
> - Support for SecureFiles LOBs
> - Ensure Table Rows in the Primary Database Can Be Uniquely Identified

C.10.1 Unsupported Tables as a Result of Partitioning

Logical standby does not support tables that use system partitioning or reference partitioning.

When possible, the ATTRIBUTES column of the DBA_LOGSTDBY_UNSUPPORTED view displays the reason why a table is not supported by SQL Apply. The ATTRIBUTES column may be NULL if the table structure itself is not supported by SQL Apply (for example, the table is system-partitioned).

C.11 PL/SQL Supplied Packages Considerations

Keep these considerations in mind regarding supported and unsupported 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.11.1 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_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 Supported Datatypes in a Logical Standby Database. Examples of such packages are DBMS_LOB, DBMS_SQL, and DBMS_TRANSACTION.

Oracle Data Guard logical standby supports replication of actions performed through the following packages: DBMS_DDL, DBMS_FGA, SDO_META, DBMS_REDACT, DBMS_REDEFINITION, DBMS_RLS, DBMS_SQL_TRANSLATOR, DBMS_XDS, DBMS_XMLINDEX and DBMS_XMLSCHEMA.

To identify which packages are supported in logical standby, you can query the DBA_LOGSTDBY_PLSQL_SUPPORT view. For example, you can run the following query to find out which packages are supported in a generic logical standby:

```
SQL> SELECT OWNER, PKG_NAME FROM DBA_LOGSTDBY_PLSQL_SUPPORT -
> where support_level = 'ALWAYS';
```

To identify which packages are supported in the context of rolling upgrades done using the DBMS_ROLLING package, you can query the DBA_LOGSTDBY_PLSQL_SUPPORT view, as follows:

```
SQL> SELECT OWNER, PKG_NAME FROM DBA_LOGSTDBY_PLSQL_SUPPORT -
> where support_level = 'DBMS_ROLLING';
```

C.11.2 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, and DBMS_AQ.

Additionally, the DBMS_RESOURCE_MANAGER package is not supported for physical standby rolling upgrades.

C.11.2.1 Support for DBMS_JOB

Specific support for DBMS_JOB has been provided. Jobs created on the primary database are replicated on the standby database, but are not run as long as the standby maintains its standby role.

In the event of a switchover or failover, jobs scheduled on the original primary database automatically begin running on the new primary database.

You can also create jobs at the logical standby. These jobs only run as long as the logical standby maintains it standby role.

C.11.2.2 Support for DBMS_SCHEDULER

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 Oracle Database 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, so 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, except in the context of a rolling upgrade done using the `DBMS_ROLLING` PL/SQL package. 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, to run jobs that need to modify unmaintained tables, set the database guard to `STANDBY`. (It is not possible to use the `ALTER SESSION DISABLE GUARD` statement inside a PL/SQL block and have it take effect.)

C.11.3 Handling XML and XDB PL/SQL Packages in Logical Standby

Logical Standby supports `XMLType` data for all storage models, with some compatibility requirements.

The requirements are as follows:

- `XMLType` stored in `CLOB` format requires that the primary database be run at a compatibility setting of 11.0 or higher (`XMLType` stored as `CLOB` is deprecated as of Oracle Database 12*c* Release 1 (12.1).)
- `XMLType` stored in object-relational format or as binary XML requires that the primary database be running Oracle Database 11*g* Release 2 (11.2.0.3) or higher with a redo compatibility setting of 11.2.0.3 or higher

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 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 The DBMS_XMLSCHEMA Schema through Compensating for Ordering Sensitive Unsupported PL/SQL for more information about dealing with these unsupported procedures.

The following packages contain unsupported procedures:

- `DBMS_XMLSCHEMA` (Supported if compatibility is set to 12.0.0 or higher.)
- `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.11.3.1 The DBMS_XMLSCHEMA Schema

Certain 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 Dealing With Unsupported PL/SQL Procedures through Compensating for Ordering Sensitive Unsupported PL/SQL provide more information on the alternatives available for dealing with these unsupported procedures:

- `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 do not get replicated:

- `GENERATEBEAN`

The following procedures and functions do not generate redo and therefore do not stop Logical Standby:

- `GENERATESCHEMAS`
- `GENERATESCHEMA`

> **Note:**
>
> As of Oracle Database 12*c* Release 1 (12.1), the `GENERATESCHEMAS` and `GENERATESCHEMA` procedures are deprecated.

C.11.3.2 The DBMS_XMLINDEX Package

All procedures in the `DBMS_XMLINDEX` package are supported except for these.

- `DBMS_XMLINDEX.REGISTERPARAMETER`
- `DBMS_XMLINDEX.MODIFYPARAMETER`
- `DBMS_XMLINDEX.DROPPARAMETER`

C.11.3.3 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.11.3.4 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"."REGISTERPARAMETER" (
   "NAME" => 'myIndexParam',
 "PARAMETER" => 'PATH TABLE

ORA-16265: Unsupported PL/SQL procedure encountered
begin
"XDB"."DBMS_XMLSCHEMA"."REGISTERPARAMETER" (
   "NAME" => 'myIndexParam',
 "PARAMETER" => 'PATH TABLE

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.11.3.5 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`. The `DBMS_XMLSCHEMA.copyEvolve` 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.12 Unsupported Tables

It is important to identify unsupported database objects on the primary database before you create a logical standby database.

This is because changes made to unsupported data types and tables on the primary database are automatically skipped by SQL Apply on the logical standby database. Moreover, no error message is 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 are implicitly maintained by SQL Apply. However, if you put a user-defined table in `SYSTEM`, then it is not 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 returns all schemas that are considered to be internal. User tables placed in these schemas are not replicated on a logical standby database and do not show up in the `DBA_LOGSTDBY_UNSUPPORTED` view. Tables in these schemas that are created by Oracle are 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 are not maintained by SQL Apply because of unsupported data types:

```
SQL> SELECT DISTINCT OWNER,TABLE_NAME FROM DBA_LOGSTDBY_UNSUPPORTED -
> 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 -
> 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:**
>
> For the queries shown in this section, if you are working in a multitenant container database (CDB) environment, then many DBA views have analogous CDB views that you should use instead. For example, you would query the `CDB_LOGSTDBY_SKIP` view instead of the `DBA_LOGSTDBY_SKIP` view.

C.12.1 Unsupported Tables During Rolling Upgrades

Before you perform a rolling upgrade, determine whether any of the tables involved contain data types that are unsupported on logical standby databases.

To do this, you can query either the `DBA_LOGSTDBY_UNSUPPORTED` view or the `DBA_ROLLING_UNSUPPORTED` view, depending on the type of rolling upgrade being performed.

If you are performing a rolling upgrade using the `DBMS_ROLLING` PL/SQL package, as described in Using DBMS_ROLLING to Perform a Rolling Upgrade , then query the `DBA_ROLLING_UNSUPPORTED` view.

If you are not using the `DBMS_ROLLING` package, but are instead following the manual process outlined in Using SQL Apply to Upgrade the Oracle Database , then query the `DBA_LOGSTDBY_UNSUPPORTED` view.

A rolling upgrade performed using `DBMS_ROLLING` supports more object types than a manual rolling upgrade operation. For example, only upgrades performed with `DBMS_ROLLING` support queue tables. Additionally, a rolling upgrade performed using `DBMS_ROLLING` also supports more PL/SQL packages.

> **See Also:**
>
> - Using SQL Apply to Upgrade the Oracle Database for more information about performing manual rolling upgrades
> - Using DBMS_ROLLING to Perform a Rolling Upgrade for more information about performing rolling upgrades using the `DBMS_ROLLING` PL/SQL package
> - Additional PL/SQL Package Support Available Only in the Context of DBMS_ROLLING Upgrades for information about PL/SQL package support available only in the context of `DBMS_ROLLING` upgrades
> - *Oracle Database PL/SQL Packages and Types Reference* for a description of the `DBMS_ROLLING` PL/SQL package
> - *Oracle Database Reference* for complete information about views

C.12.2 Unsupported Tables As a Result of DML Performed In a PL/SQL Function

If, during an insert or update DML operation on a supported table, an out-of-line column (LOB, XMLType, or ADT) is modified through a PL/SQL function and that function in turn performs DML on another table in the course of its execution, then the redo patterns generated are unsupported by LogMiner.

As a result, redo for such a workload cannot be reliably mined using LogMiner.

C.13 Skipped SQL Statements on a Logical Standby Database

By default, certain SQL statements are automatically skipped by SQL Apply.

The affected statements are as follows:

```
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
PURGE DBA_RECYCLEBIN
PURGE INDEX
SET CONSTRAINTS
SET ROLE
SET TRANSACTION
```

All other SQL statements executed on the primary database are applied to the logical standby database.

C.14 DDL Statements Supported by a Logical Standby Database

The `DBMS_LOGSTDBY.SKIP` procedure has several optional keywords.

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. 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 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`
	`REVOKE`
	`ANALYZE TABLE`
	`ANALYZE INDEX`
	`ANALYZE CLUSTER`

`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`

`DIRECTORY`	`CREATE DIRECTORY` `DROP DIRECTORY`
`DML`	Includes DML statements on a table (for example: `INSERT`, `UPDATE`, and `DELETE`)

Table C-1 (Cont.) Values for stmt Parameter of the DBMS_LOGSTDBY.SKIP procedure

Keyword	Associated SQL Statements
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 SEGMENT	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

Table C-1 (Cont.) Values for stmt Parameter of the DBMS_LOGSTDBY.SKIP procedure

Keyword	Associated SQL Statements
SYNONYM	CREATE PUBLIC SYNONYM CREATE SYNONYM DROP PUBLIC SYNONYM DROP SYNONYM
SYSTEM AUDIT	AUDIT *SQL_statements* NOAUDIT *SQL_statements*
TABLE	CREATE TABLE ALTER 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:
>
> - 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

C.14.1 DDL Statements that Use DBLINKS

SQL Apply may not correctly apply DDL statements that reference a database link.

An example of such a statement is as follows:

```
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, then use the `DBMS_LOGSTDBY.INSTANTIATE_TABLE` procedure for the table being created, and then restart SQL APPLY operations.

C.14.2 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.

C.15 Distributed Transactions and XA Support

You can perform distributed transactions using a couple different methods.

- Modify tables in multiple databases in a coordinated manner using database links.

- Use the XA interface, as exposed by the `DBMS_XA` package in supplied PL/SQL packages or via OCI or JDBC libraries. The XA interface implements X/Open Distributed Transaction Processing (DTP) architecture.

Changes made to the primary database during a distributed transaction using either of these two methods are replicated to the logical standby database.

However, the distributed transaction state is not replicated. The logical standby database does not inherit the in-doubt or prepared state of such a transaction, and it does not replicate the changes using the same global transaction identifier used at the primary database for the XA transactions. As a result, if you fail over to a logical standby database before committing a distributed transaction, the changes are rolled back at the logical standby. This rollback occurs even if the distributed transaction on the primary database is in a prepared state and has successfully completed the first phase of the two-phased commit protocol. Switchover operations wait for all active distributed transactions to complete, and are not affected by this restriction.

XA transactions can be performed in two ways:

- tightly coupled, where different XA branches share locks
- loosely coupled, where different XA branches do not share locks

Replication of changes made by loosely coupled XA branches is supported regardless of the `COMPATIBLE` parameter value. Replication of changes made by tightly coupled branches on an Oracle RAC primary (introduced in 11g Release 1) is supported only with `COMPATIBLE=11.2` or higher.

C.16 Support for SecureFiles LOBs

SecureFiles LOBs are supported when the database compatibility level is set to 11.2 or higher.

Transparent Data Encryption and data compression can be enabled on SecureFiles LOB columns at the primary database.

Deduplication of SecureFiles LOB columns and SecureFiles Database File System (DBFS) operations are fully supported. Fragment operations are only supported via Extended Datatype Support (EDS).

If SQL Apply encounters redo generated by unsupported operations, it stops with an `ORA-16211: Unsupported record found in the archived redo log` error. To continue, add a skip rule for the affected table using `DBMS_LOGSTDBY.SKIP` and restart SQL Apply.

C.17 Support for Database File System (DBFS)

The Database File System (DBFS) creates a standard file system interface on top of files and directories that are stored in database tables, which makes it easier for you to access and manage files stored in the database.

Logical standby supports the Database File System (DBFS). See *Oracle Database SecureFiles and Large Objects Developer's Guide* for more information about DBFS.

C.18 Character Set Considerations

There are considerations to keep in mind regarding character sets.

- It is not supported to have a Data Guard configuration in which the primary database and logical standby database have different character sets.
- Configurations in which a multitenant container database (CDB) has a mixed character set are only supported when using `DBMS_ROLLING` for rolling upgrades. A mixed character set means that the `CDB$ROOT` and one or more of the CDB's pluggable databases (PDBs) have different character sets.

C.19 Additional PL/SQL Package Support Available Only in the Context of DBMS_ROLLING Upgrades

Replication of certain packages is available only in the context of rolling upgrades performed using the `DBMS_ROLLING` package.

The affected packages are as follows:

- DBFS
  - `DBMS_DBFS_CONTENT_ADMIN`
  - `DBMS_DBFS_SFS`
  - `DBMS_DBFS_SFS_ADMIN`
- Lightweight Security
  - `XS_ACL`
  - `XS_DATA_SECURITY`
  - `XS_NAMESPACE`
  - `XS_PRINCIPAL`
  - `XS_ROLESET`
  - `XS_SECURITY_CLASS`
- Oracle Streams Advanced Queuing (AQ)
  - `DBMS_AQ`
  - `DBMS_AQJMS`
  - `DBMS_AQADM` (except for the following procedures: `SCHEDULE_PROPAGATION`, `RECOVER_PROPAGATION`, `UNSCHEDULE_PROPAGATION`, `ALTER_PROPAGATION_SCHEDULE`, `ENABLE_PROPAGATION_SCHEDULE`, and `DISABLE_PROPAGATION_SCHEDULE`)
- Oracle Text
  - `CTX_ADM`
  - `CTX_ANL`
  - `CTX_CLS`
  - `CTX_DDL`
  - `CTX_DOC`
  - `CTX_ENTITY`
  - `CTX_OUTPUT`
  - `CTX_QUERY`

  - CTX_THES
  - CTX_TREE
- Scheduler
  - DBMS_SCHEDULER
- XDB-related
  - DBMS_RESCONFIG
  - DBMS_XDB_CONFIG (Certain procedures are not supported. See *Oracle XML DB Developer's Guide* for more information.)
  - DBMS_XDB_REPOS
  - DBMS_XDBRESOURCE
  - DBMS_XDB_VERSION
  - DBMS_XDBZ (Certain procedures are not supported. See *Oracle XML DB Developer's Guide* for more information.)

See Also:

- Using DBMS_ROLLING to Perform a Rolling Upgrade
- *Oracle Database Real Application Security Administrator's and Developer's Guide* for more information about the Lightweight Security packages
- *Oracle Database PL/SQL Packages and Types Reference* for more information about DBMS_SCHEDULER, XDB-related, and DBFS-related packages

D

Oracle 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 (Oracle RAC) multiple-instance databases.

This appendix summarizes the configuration requirements and considerations that apply when using Oracle Data Guard with Oracle RAC databases. It contains the following sections:

D.1 Configuring Standby Databases in an Oracle RAC Environment

You can configure a standby database to protect a primary database using Oracle Real Application Clusters (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	No
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. As of Oracle Database 12c release 2 (12.2.0.1) you can also perform multi-instance redo apply.

D.1.1 Setting Up Multi-Instance Redo Apply

As of Oracle Database 12c Release 2 (12.2.0.1), a new `INSTANCES [ ALL | integer]` clause is available on the SQL `ALTER DATABASE RECOVER MANAGED STANDBY DATABASE` command.

It has the following restrictions:

- The clause is applicable only for Oracle Real Application Clusters (Oracle RAC) or Oracle RAC One Node databases.
- Block Change tracking is not supported.
- In-Memory column store is not supported with multi-instance redo apply in an Active Data Guard (ADG) environment.

The `ALL` option causes redo apply to run on all instances in an Oracle RAC standby database that are in an open or mounted state at the time recovery is started. .All instances must be in the same state — either open or mounted. A mix of states is not allowed.

The *`integer`* option restricts the number of instances that redo apply uses to the number you specify. For integer, specify an integer value from 1 to the number of instances in the standby database. The database chooses the instances on which to perform Redo Apply; you cannot specify particular instances.

The `V$RECOVERY_PROGRESS` view is only populated on the instance where recovery was started (where the MRP0 process resides).

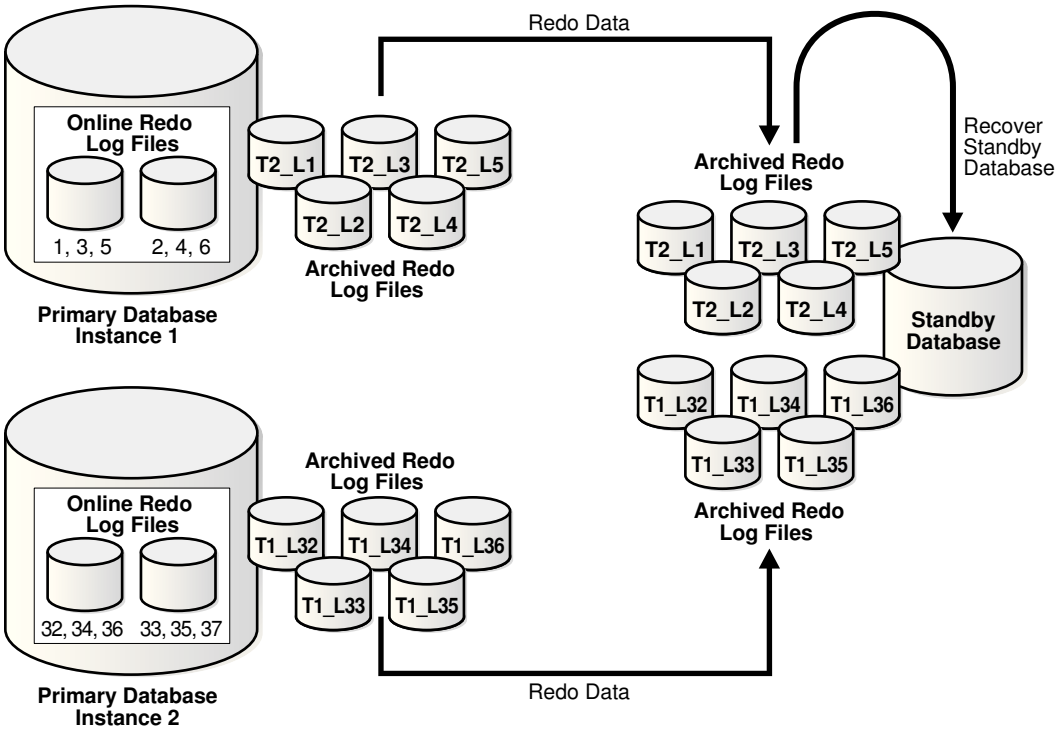
If you omit the `INSTANCES` clause, then recovery happens on only one instance where the command was issued.

Because recovery processes ship redo among instances, redo apply performance is directly related to network bandwidth and latency.

D.1.2 Setting Up a Multi-Instance Primary with a Single-Instance Standby

This figure 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.

Although Figure D-1 does not show standby redo logs, it is a best practice to configure standby redo logs at the standby for both instances of the primary. The redo from the primary online redo log files at Instance 1 and Instance 2 would then be received first in the standby redo logs for Instance 1 and Instance 2, respectively, and then archived.

To Configure a Primary Database in an Oracle RAC Environment

Before you create a standby database you must first ensure the primary database is properly configured. To do so, you must perform some preparatory steps, after which the database is prepared to serve as the primary database for one or more standby databases.

To Configure a Single Instance Standby Database

To specify the location of the archived redo log files and standby redo log files, define the `LOG_ARCHIVE_DEST_`*n* and `LOG_ARCHIVE_FORMAT` parameters.

> **See Also:**
>
> - Managing Standby Redo Logs for more information about standby redo logs
> - Creating a Physical Standby Database for information about configuring a primary database when creating a physical standby database.
> - Creating a Logical Standby Database for information about configuring a primary database when creating a logical standby database.

D.1.3 Setting Up Oracle RAC Primary and Standby Databases

An Oracle RAC primary database must be set up to send redo data to an Oracle RAC standby database, and an Oracle RAC standby database must be set up to receive redo data.

D.1.3.1 Configuring an Oracle RAC Standby Database to Receive Redo Data

These steps describe how 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 Oracle ASM instance. See Managing Standby Redo Logs 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.

D.1.3.2 Configuring an Oracle RAC Primary Database to Send Redo Data

Configure each instance of the Oracle RAC primary database to send its redo data to the Oracle RAC standby database. These are the best practices that Oracle recommends you follow when you configure 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.

Configuring an Oracle Database to Send Redo Data describes how to configure an Oracle database instance to send redo data to another database.

D.2 Configuration Considerations in an Oracle RAC Environment

Oracle Real Application Clusters (Oracle RAC) have specific Oracle Data Guard configuration requirements regarding the format for archived redo log filenames, and for data protection modes.

See the following topics:

- Format for Archived Redo Log Filenames
- Data Protection Modes

D.2.1 Format for Archived Redo Log Filenames

The format for archived redo log filenames is in the form of log_*%parameter*.

The *%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.

Table D-1 (Cont.) Directives for the LOG_ARCHIVE_FORMAT Initialization Parameter

Directives	Description
%S	Log file sequence number, zero filled.
%r	Resetlogs ID.
%R	Resetlogs ID, zero filled.

For example:

```
LOG_ARCHIVE_FORMAT = log%d_%t_%s_%r.arc
```

The thread parameters %t or %T are mandatory for Oracle RAC to uniquely identify the archived redo log files with the `LOG_ARCHIVE_FORMAT` parameter.

D.2.2 Data Protection Modes

If any instance of an Oracle RAC primary database loses connectivity with a standby database, all other primary database instances stop sending redo to the standby database for the number of seconds specified on the `LOG_ARCHIVE_DEST_`*n* `REOPEN` attribute, after which all primary database instances attempt to reconnect to the standby database.

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, then the instance loses connectivity and is shut down. Other instances in an Oracle RAC configuration that still have connectivity to the standby destinations recover the lost instance and continue sending redo to their standby destinations. Only when every instance in an Oracle RAC configuration loses connectivity to the last standby destination is the primary database shut down.

- Maximum availability and maximum performance configurations

  Other instances in an Oracle RAC configuration that still have connectivity to the standby destination recover the lost instance and continue sending redo to their standby destinations. When every instance in an Oracle RAC configuration loses connectivity to the standby destination, the primary database continues operation in maximum performance mode. The maximum performance mode ensures very minimal data loss except when the entire standby fails.

  The maximum availability protection mode ensures zero data loss except in the case of certain double faults, such as failure of the primary database after the failure of all standby databases.

E

Creating a Standby Database with Recovery Manager

These topics describe how to use Oracle Recovery Manager to create a standby database.

- Prerequisites
- Overview of Standby Database Creation with RMAN
- Using the DUPLICATE Command to Create a Standby Database

E.1 Prerequisites

Before you create a standby database with RMAN, you should be familiar with database duplication.

Because you use the `DUPLICATE` command to create a standby database with RMAN, you should also familiarize yourself with the `DUPLICATE` command .

Also familiarize yourself with how to create a standby database *before* you attempt to create a standby database with RMAN.

> **See Also:**
>
> - *Oracle Database Backup and Recovery User's Guide* for information about database duplication
> - *Oracle Database Backup and Recovery Reference* for a description of the RMAN `DUPLICATE` command
> - Creating a Physical Standby Database
> - Creating a Logical Standby Database

E.2 Overview of Standby Database Creation with RMAN

Read this section for information about the purpose and basic concepts involved in standby database creation with RMAN.

E.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.

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.

E.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. Therefore, do not register the standby database with your recovery catalog.

E.2.2.1 Active Database and Backup-Based Duplication

When you are using RMAN to create a standby, you must choose between active and backup-based duplication.

If you specify `FROM ACTIVE DATABASE`, then RMAN copies the data files 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 data files 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 data files unless you execute the `SET UNTIL` command.

E.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 you use RMAN in an Oracle Data Guard environment, connect it to a recovery catalog. The recovery catalog can store the metadata for all primary and standby

databases in the environment. Do not explicitly register the standby database in the recovery catalog.

A database in an Oracle 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 an Oracle Data Guard environment.

> **See Also:**
>
> *Oracle Database Backup and Recovery User's Guide* for a conceptual overview of RMAN operation in an Oracle Data Guard environment

E.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.

For RMAN to recover the standby database after creating it, 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 you want RMAN to 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.

E.2.2.3.1 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.
- For the standby redo log file names on the standby database to be the same as the primary redo log file names, 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.

E.2.2.4 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.

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 Oracle Data Guard to ship logs.

E.3 Using the DUPLICATE Command to Create a Standby Database

The procedure for creating a standby database is basically identical to the RMAN duplication procedure.

See *Oracle Database Backup and Recovery User's Guide* for information about the RMAN duplication procedure.

E.3.1 Using Active Database Duplication to Create a Standby Database or Far Sync Instance

You can use active database duplication to create either a standby database or a far sync instance.

Creating a Standby Database with Active Database Duplication

To use the RMAN `DUPLICATE` command to create a standby database from files that are active in the primary database, you must specify both the `FOR STANDBY` and `FROM ACTIVE DATABASE` options. You can specify other options as well, such as `DORECOVER`. For example:

```
DUPLICATE TARGET DATABASE FOR STANDBY FROM ACTIVE DATABASE DORECOVER;
```

Creating a Far Sync Instance with Active Database Duplication

As of Oracle Database 12*c* Release 12.2 (12.2.0.1), you can also use the RMAN `DUPLICATE` command to create a far sync instance from files that are active in the primary database. To do so, substitute `FARSYNC` in place of `STANDBY` on the command line (do not specify the `DORECOVER` option; it is not allowed for far sync instances). For example:

```
DUPLICATE TARGET DATABASE FOR FARSYNC FROM ACTIVE DATABASE;
```

Steps to Create a Standby Database or Far Sync Instance Using Active Database Duplication

The following steps create a standby database from active database files, but you could also use these steps to create a far sync instance from active database files. The steps assume that the standby host (or far sync instance), and primary database host have the same directory structure.

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 which is overwritten during the duplicate operation.

2. Decide how to provide names for the standby control files, data files, 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 are 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 Oracle Data Guard can identify this database as being different from the primary. Optionally, specify the `DORECOVER` option to recover the

database after standby creation (DORECOVER is not a valid option for far sync instances).

```
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_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.

If you were creating a far sync instance rather than a standby, the command would be the same except that STANDBY is replaced with FARSYNC, as follows:

```
DUPLICATE TARGET DATABASE
  FOR FARSYNC
  FROM ACTIVE DATABASE
  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_SERVER="inst1" COMMENT "Is primary"
  NOFILENAMECHECK;
```

> **Note:**
>
> If the primary is an Oracle Real Application Clusters (Oracle RAC) database, and the target standby database or far sync instance is going to be a single-instance Oracle database, then add the command SET CLUSTER_DATABASE="FALSE" to the previous RMAN examples.

E.3.2 Creating a Standby Database with Backup-Based Duplication

You can use backup—based duplication to create either a standby database or a far sync instance.

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. You can specify other options as well, such as DORECOVER. For example:

```
DUPLICATE TARGET DATABASE FOR STANDBY BACKUP LOCATION '+DATA/BACKUP' DORECOVER;
```

Creating a Far Sync Instance with Backup-Based Duplication

As of Oracle Database 12*c* Release 12.2 (12.2.0.1), you can also use backup-based duplication to create a Data Guard far sync instance using the RMAN DUPLICATE

command. To do so, substitute `FARSYNC` in place of `STANDBY` on the command line (do not specify the `DORECOVER` option; it is not allowed for far sync instances). For example:

```
DUPLICATE TARGET DATABASE FOR FARSYNC BACKUP LOCATION '+DATA/BACKUP';
```

Steps to Create a Standby Database or Far Sync Instance from Backups:

The following steps create a standby database from backup files, but you could also use these steps to create a far sync instance from backup files. The steps assume that the standby host (or far sync instance), and primary database host have the same directory structure.

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, data files, 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 are 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. Optionally, specify the `DORECOVER` option to recover the database after standby creation (`DORECOVER` is not a valid option for far sync instances).

   ```
   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_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.

F

Setting Archive Tracing

The Oracle database uses the LOG_ARCHIVE_TRACE initialization parameter to enable and control the generation of comprehensive trace information for log archiving and redo transport activity.

The additional tracing that is output when LOG_ARCHIVE_TRACE is set to a nonzero value can appear in trace files for an archive process, RFS process, LGWR process, SYNC process, ASYNC process, foreground process, MRP process, recovery process, log apply process, startup process, shutdown process, and other processes that use redo transport services. Tracing information is written to the Automatic Diagnostic Repository.

> **See Also:**
>
> - *Oracle Database Reference* for a complete description of the LOG_ARCHIVE_TRACE initialization parameter and its valid values
> - *Oracle Database Administrator's Guide* for more information about the Automatic Diagnostic Repository

F.1 Setting the LOG_ARCHIVE_TRACE Initialization Parameter

To enable, disable, or modify the LOG_ARCHIVE_TRACE parameter, you issue an ALTER SYSTEM SQL statement.

The format for the LOG_ARCHIVE_TRACE parameter is as follows, where *trace_level* is an integer:

```
LOG_ARCHIVE_TRACE=trace_level
```

The following is an example of setting the LOG_ARCHIVE_TRACE parameter:

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_TRACE=15;
```

You can combine tracing levels by setting the value of the LOG_ARCHIVE_TRACE parameter to the sum of the individual levels. For instance, in the previous example, setting the LOG_ARCHIVE_TRACE parameter to a value of 15 sets trace levels 1, 2, 4, and 8.

For a complete list and description of valid LOG_ARCHIVE_TRACE values, see *Oracle Database Reference*.

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 remote file server (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
```

G

Performing Role Transitions Using Old Syntax

Prior to Oracle Database 12*c* Release 1 (12.1), the procedures for performing switchovers and failovers to a physical standby database were different.

These procedures are still supported, but Oracle recommends you use the new procedures described in "Role Transitions Involving Physical Standby Databases".

If you are using a release prior to Oracle Database 12*c* Release 1 (12.1), then you must use the old procedures.

The following topics are discussed:

- SQL Syntax for Role Transitions Involving Physical Standbys
- Role Transitions Involving Physical Standby Databases
- Troubleshooting Switchovers to Physical Standby Databases

G.1 SQL Syntax for Role Transitions Involving Physical Standbys

Oracle Database 12*c* Release 1 (12.1) introduces new SQL syntax for performing switchover and failover operations to a physical standby database.

Oracle Database 12*c* Release 1 (12.1) introduces new SQL syntax for performing switchover and failover operations to a physical standby database. Do not mix syntax from the old procedures (described in this topic) and the new procedures (described in Role Transitions), unless you are specifically directed to do so.

Pre-12c Role Transition Syntax for Physical Standby Databases	12c Role Transition Syntax for Physical Standby Databases

To switchover to a physical standby database, on the primary database:
`SQL> ALTER DATABASE COMMIT TO SWITCHOVER TO PHYSICAL STANDBY;`
On the physical standby database:
`SQL>ALTER DATABASE COMMIT TO SWITCHOVER TO PRIMARY;`
	To switchover to a physical standby database:
	`SQL> ALTER DATABASE SWITCHOVER TO` *`target_db_name`* `[FORCE] [VERIFY];`

Pre-12c Role Transition Syntax for Physical Standby Databases	12c Role Transition Syntax for Physical Standby Databases

To failover to a physical standby database, (step 6 and step 8 in "Performing a Failover to a Physical Standby Database Using Old Syntax"):
`SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE FINISH;`
and
`SQL> ALTER DATABASE COMMIT TO SWITCHOVER TO PRIMARY;`
	To failover to a physical standby database, the following statement replaces the two statements previously required:
	`SQL> ALTER DATABASE FAILOVER TO` *`target_db_name`*`;`

> **See Also:**
>
> - *Oracle Database SQL Language Reference* for more information about SQL syntax

G.1.1 New Features When Using the Old Syntax

As of Oracle Database 12*c* Release 1 (12.1), the `WITH SESSION SHUTDOWN` clause is no longer needed to kill active SQL sessions.

You can issue the following statement to automatically kill active SQL sessions:

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

Additionally, when you perform a switchover from an Oracle RAC primary database to a physical standby database, it is no longer necessary to shut down all but one primary database instance. All the instances are shut down automatically after the switchover is complete.

G.2 Role Transitions Involving Physical Standby Databases

The SQL syntax to perform switchovers and failovers to a physical standby database was different in releases prior to Oracle Database 12*c* Release 1 (12.1).

The following are the procedures that must be used if you are running a release prior to 12.1:

- Performing a Switchover to a Physical Standby Database Using Old Syntax
- Performing a Failover to a Physical Standby Database Using Old Syntax

> **See Also:**
>
> Role Transitions for information about how to prepare for switchovers and failovers

G.2.1 Performing a Switchover to a Physical Standby Database Using Old Syntax

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

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

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 Redo Transport Services for information about configuring and monitoring redo transport.

2. Issue the following SQL statement on the primary database to switch it to the standby role: 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;
   ```

   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.

3. Mount the former primary database. For example: For example:

   ```
   SQL> STARTUP MOUNT;
   ```

   At this point in the switchover process, the original primary database is a physical standby database.

4. Query the `SWITCHOVER_STATUS` column of the `V$DATABASE` view on the standby database to verify that the switchover target is ready to be switched to the primary role. 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`.

5. Issue the following SQL statement on the target physical standby database to switch it to the primary role:

```
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 step 4 returned a value of `TO PRIMARY`.

6. Open the new primary database:

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

7. Start Redo Apply on the new physical standby database:

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

8. Restart Redo Apply if it has stopped at any of the other physical standby databases in your Oracle Data Guard configuration:

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

G.2.2 Performing a Failover to a Physical Standby Database Using Old Syntax

These steps describe how to perform a failover to a physical standby database.

1. If possible, mount the primary database and flush any unsent archived and current redo from the primary database to the standby database. If this operation is successful, a zero data loss failover is possible even if the primary database is not in a zero data loss data protection mode.

   Ensure that Redo Apply is active at the target standby database.

   Mount, but do not open, the primary database. If the primary database cannot be mounted, go to step 2.

   Issue the following SQL statement at the primary database:

```
SQL> ALTER SYSTEM FLUSH REDO TO target_db_name;
```

   For *target_db_name*, specify the `DB_UNIQUE_NAME` of the standby database that is to receive the redo flushed from the primary database.

   This statement flushes any unsent redo from the primary database to the standby database, and waits for that redo to be applied to the standby database.

   If this statement completes without any errors, go to step 5. If the statement completes with any errors, or if it must be stopped because you cannot wait any longer for the statement to complete, continue with step 2.

2. Verify that the standby database has the most recently archived redo log file for each primary database redo thread.

To do this, query the V$ARCHIVED_LOG view on the target standby database to obtain the highest log sequence number for each redo thread.

For example:

```
SQL> SELECT UNIQUE THREAD# AS THREAD, MAX(SEQUENCE#) -
> OVER (PARTITION BY thread#) AS LAST from V$ARCHIVED_LOG;

    THREAD       LAST
---------- ----------
         1        100
```

If possible, copy the most recently archived redo log file for each primary database redo thread to the standby database if it does not exist there, and register it. This must be done for each redo thread.

For example:

```
SQL> ALTER DATABASE REGISTER PHYSICAL LOGFILE 'filespec1';
```

3. Query the V$ARCHIVE_GAP view on the target standby database 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 sequence numbers 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 at the target standby database. This must be done for each redo thread.

   For example:

   ```
   SQL> ALTER DATABASE REGISTER PHYSICAL LOGFILE 'filespec1';
   ```

4. Repeat step 3 until all gaps are resolved. (The query executed in step 3 displays information for the highest gap only, so after resolving a gap, you must repeat the query until no more rows are returned.)

   If, after performing steps 2 through step 4, 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), then some data loss will occur during the failover.

5. Issue the following SQL statement on the target standby database to stop Redo Apply:

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

6. Finish applying all received redo data:

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

   If this statement completes without any errors, then proceed to step 7.

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

If there is a redo gap that was not resolved in step 3 and step 4, then you receive an error stating that there is a redo gap.

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

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

Proceed to step 9 when the `ACTIVATE` statement completes.

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

   To do this, 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.

8. Issue the following SQL statement on the target standby database to switch the physical standby to the primary role:

   ```
   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 a value of `TO PRIMARY`.

9. Open the new primary database:

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

10. Oracle recommends that you now back up the new primary database.

11. Restart Redo Apply if it has stopped at any of the other physical standby databases in your Oracle Data Guard configuration:

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

12. 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 Converting a Failed Primary Into a Standby Database Using Flashback Database or Converting a Failed Primary into a Standby Database Using RMAN Backups, or it can be re-created as a physical standby database from a backup of the new primary database using the method described in Step-by-Step Instructions for Creating a Physical Standby Database.

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

G.3 Troubleshooting Switchovers to Physical Standby Databases

These are some of the problems that can occur during a switchover to a physical standby database.

> **Note:**
>
> The following troubleshooting topics apply only when you are performing switchovers and failovers to a physical standby database using procedures available in releases prior to Oracle Database 12c Release 1 (12.1).

- Switchover Fails Because Redo Data Was Not Transmitted
- Switchover Fails with the ORA-01102 Error
- Redo Data Is Not Applied After Switchover
- Roll Back After Unsuccessful Switchover and Start Over

G.3.1 Switchover Fails Because Redo Data Was Not Transmitted

If a 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 is applied automatically. Query the `SWITCHOVER_STATUS` column in the `V$DATABASE` view. A switchover to the primary role is now possible if the `SWITCHOVER_STATUS` column returns `TO PRIMARY` or `SESSIONS ACTIVE`:

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

SWITCHOVER_STATUS
-----------------
TO PRIMARY
1 row selected
```

See Views Relevant to Oracle Data Guard 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 Performing a Switchover to a Physical Standby Database Using Old Syntax and try again to switch the target standby database to the primary role.

G.3.2 Switchover Fails with the ORA-01102 Error

These are some of the possible causes, and solutions, if you receive an 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 an `ORA-01102 cannot mount database in EXCLUSIVE mode` error.

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 (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.

G.3.3 Redo Data Is Not Applied After Switchover

If archived redo log files are not applied to the new standby database after the switchover, it may be 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 to enable the standby site to automatically add new data files that are created at the primary site.

G.3.4 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.

Take the following steps:

1. Shut down and mount the new standby database (old primary).
2. Start Redo Apply on the new standby database.
3. Verify that the new standby database is ready to be switched back to the primary role. Query the SWITCHOVER_STATUS column of the V$DATABASE view on the new 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 new standby database is ready to be switched to the primary role. Continue to query this column until the value returned is either TO PRIMARY or SESSIONS ACTIVE.
4. Issue the following statement to convert the new standby database back to the primary role:

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

   If this statement is successful, the database runs in the primary database role, and you do not need to perform any more steps.

   If this statement is unsuccessful, then continue with Step 5.
5. 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 reverts the new standby database back to the primary role.
6. Shut down the original physical standby database.
7. 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. Create a Control File for the Standby Database describes how to create a physical standby control file.
8. Restart the original physical standby instance.

If this procedure is successful and archive gap management is enabled, then the FAL processes 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 Manual Gap Resolution for information about archive gap management and Setting Archive Tracing for information about locating the trace files.

9. Try the switchover again.

   At this point, the Oracle 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).

H

Using the ALTERNATE Attribute to Configure Remote Alternate Destinations

As of Oracle Database 12*c* Release 2 (12.2), the preferred method for creating alternate log archive destinations for remote standby databases and far sync instances that take over if the active destination fails is to use the `GROUP` and `PRIORITY` attributes.

For local archiving locations (`LOCATION=…`), the `ALTERNATE` attribute is still used to provide high availability if the original archiving directory becomes unavailable due to disk or network issues that prevent access to the archiving location. But use of the `ALTERNATE` attribute for remote log archive destinations (`SERVICE=…`) is maintained only for backwards compatibility. The examples provided in the following sections for using this method are a follow-on for creating a far sync instance, but they also apply to cascading redo destinations.

After you perform the steps in Creating and Configuring a Far Sync Instance, the far sync instance provides zero data loss capability for the configuration to the terminal standby at a remote site over the WAN. For the configuration to remain protected, but at a reduced protection level, in the event that communication with the far sync instance is lost, you can optionally configure the terminal standby to automatically become the alternate destination. This reduces the amount of data loss by allowing Oracle Data Guard to ship redo asynchronously directly from the primary to the terminal standby, temporarily bypassing the far sync instance.

> **See Also:**
>
> - Alternate Destinations

H.1 Configuring an Alternate Destination

To configure an alternate destination, set these parameters on the primary database.

Primary Database chicago

```
LOG_ARCHIVE_DEST_STATE_2='ENABLE'

LOG_ARCHIVE_DEST_2='SERVICE=chicagoFS SYNC AFFIRM MAX_FAILURE=1
ALTERNATE=LOG_ARCHIVE_DEST_3
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) DB_UNIQUE_NAME=chicagoFS'

LOG_ARCHIVE_DEST_STATE_3='ALTERNATE'

LOG_ARCHIVE_DEST_3='SERVICE=boston ASYNC ALTERNATE=LOG_ARCHIVE_DEST_2
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) DB_UNIQUE_NAME=boston'
```

This configuration enables Oracle Data Guard to continue sending redo, asynchronously, to the terminal standby `boston` when it can no longer send the redo

directly to the far sync instance chicagoFS. When the far sync instance becomes available again, Oracle Data Guard automatically resynchronizes the far sync instance chicagoFS and returns to the original configuration in which the primary sends redo to the far sync instance and the far sync instance forwards that redo to the terminal standby. When the synchronization is complete, the alternate destination (LOG_ARCHIVE_DEST_3 in the preceding example) again becomes dormant as the alternate.

In the above case, the ALTERNATE remote destination is set up directly between two databases using asynchronous redo transport, so in the event of a failure of the far sync instance, the protection level of the configuration falls back down to maximum performance, with data loss at failover time. For more protection from system or network failures, an additional far sync instance can be configured that provides high availability for the active far sync instance.

In the following configuration, one far sync instance is the preferred active far sync instance and the other is the alternate far sync instance. Configuring an alternate far sync instance provides continued protection for the configuration if the preferred far sync instance fails for some reason, keeping the configuration at maximum availability. The primary automatically starts shipping to the alternate far sync instance if it detects a failure at the preferred far sync instance. If the preferred far sync instance then re-establishes itself, the primary switches back to the preferred far sync instance and puts the alternate far sync instance back into the alternate state.

In these types of configurations, the primary uses only one of the two far sync instances to redistribute redo at any given time.

The second high availability far sync instance would be created using the same steps as given in Creating and Configuring a Far Sync Instance, and then become the alternate to the existing far sync instance instead of the terminal standby. When complete, chicago would have the parameters configured as follows (assuming the name chicagoFS1 as the new far sync instance name).

Primary Database chicago

```
LOG_ARCHIVE_CONFIG='DG_CONFIG=(chicago,chicagoFS,chicagoFS1,boston)'

LOG_ARCHIVE_DEST_STATE_2='ENABLE'

LOG_ARCHIVE_DEST_2='SERVICE=chicagoFS SYNC AFFIRM MAX_FAILURE=1
ALTERNATE=LOG_ARCHIVE_DEST_3
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) DB_UNIQUE_NAME=chicagoFS'

LOG_ARCHIVE_DEST_STATE_3='ALTERNATE'

LOG_ARCHIVE_DEST_3='SERVICE=chicagoFS1 SYNC AFFIRM ALTERNATE=LOG_ARCHIVE_DEST_2
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE) DB_UNIQUE_NAME=chicagoFS1'
```

Primary Database boston

```
LOG_ARCHIVE_CONFIG='DG_CONFIG=(chicago,chicagoFS,chicagoFS1,boston)'
```

Oracle Data Guard can now continue to synchronously send redo to a far sync instance, maintaining the required zero data loss protection mode of maximum availability in the event that a far sync instance fails for some reason. As before, when the failed far sync instance becomes available again, Oracle Data Guard automatically resynchronizes it and returns to the original configuration in which the primary sends redo to the first far sync instance, which then forwards that redo to the terminal standby. When the synchronization is complete, the alternate destination

(LOG_ARCHIVE_DEST_3 in the preceding example) again becomes dormant as the alternate. But if both far sync instances fail, then redo is not sent to the terminal standby boston because there is no third alternate capability. That scenario can be accomplished using the GROUP and PRIORITY attributes instead of the ALTERNATE attribute.

See Also:

- ALTERNATE
- GROUP
- PRIORITY

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W