# Oracle® Data Guard

Concepts and Administration

Release 2 (9.2)

October 2002

Part No. A96653-02

ORACLE®

Oracle Data Guard Concepts and Administration, Release 2 (9.2)

Part No. A96653-02

# Contents

## 2 Getting Started with Data Guard



## 3 Creating a Physical Standby Database

## 4 Creating a Logical Standby Database



## 5 Log Transport Services

## 6 Log Apply Services

## 9 Managing a Logical Standby Database



## 10 Data Guard Scenarios

## 13 SQL Statements



## 14 Views

## B Manual Recovery



## C Standby Database Real Application Clusters Support



## D Cascaded Redo Log Destinations

## List of Examples

## List of Figures

## List of Tables

# Send Us Your Comments

**Oracle Data Guard Concepts and Administration, Release 2 (9.2)**

**Part No. A96653-02**

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

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

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

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

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

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

# Preface

A standby database is the most effective disaster recovery solution for an Oracle database because a standby database can be used to run your production system if your primary database becomes unusable. A standby database can also be used to remedy problems caused by user errors, data corruption, and other operational difficulties.

This guide describes Oracle Data Guard concepts, and helps you configure and implement standby databases.

This preface contains the following topics:



## 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.

## Documentation Accessibility

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

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

## Organization

This document contains:

### Part I, "Concepts and Administration"

#### Chapter 1, "Introduction to Oracle Data Guard"

This chapter offers a general overview of the Oracle9*i* Data Guard architecture.

#### Chapter 2, "Getting Started with Data Guard"

This chapter introduces physical, logical, and cascading standby databases.

#### Chapter 3, "Creating a Physical Standby Database"

This chapter explains how to create a physical standby database and start applying redo logs to it.

#### Chapter 4, "Creating a Logical Standby Database"

This chapter explains how to create a logical standby database and start applying redo logs to it.

### Chapter 5, "Log Transport Services"

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

### Chapter 6, "Log Apply Services"

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

### Chapter 7, "Role Management"

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

### Chapter 8, "Managing a Physical Standby Database"

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

### Chapter 9, "Managing a Logical Standby Database"

This chapter describes how to manage a logical standby database. It provides information on applying redo logs, system tuning, and tablespace management.

### Chapter 10, "Data Guard Scenarios"

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

## Part II, "Reference"

### Chapter 11, "Initialization Parameters"

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

### Chapter 12, "LOG_ARCHIVE_DEST_n Parameter Attributes"

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

### Chapter 13, "SQL Statements"

This reference chapter provides SQL statements that are useful for performing operations on a standby database.

### Chapter 14, "Views"

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

## Part III, "Appendixes and Glossary"

### Appendix A, "Troubleshooting the Standby Database"

This appendix discusses troubleshooting for the standby database.

### Appendix B, "Manual Recovery"

This appendix describes managing a physical standby database in manual recovery mode. It provides instructions for manually resolving archive gaps and renaming standby files not captured by conversion parameters.

### Appendix C, "Standby Database Real Application Clusters Support"

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

### Appendix D, "Cascaded Redo Log Destinations"

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

### Appendix E, "Sample Disaster Recovery ReadMe File"

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

### Glossary

# Related Documentation

Every reader of *Oracle Data Guard Concepts and Administration* should have read:

- The beginning of *Oracle9i Database Concepts*, which provides an overview of the concepts and terminology related to the Oracle database server and a foundation for the more detailed information in this guide. The rest of *Oracle9i Database Concepts* explains the Oracle architecture and features in detail.
- The chapters in the *Oracle9i Database Administrator's Guide* that deal with managing the control file, online redo logs, and archived redo logs.

You will often need to refer to the following guides:

- *Oracle9i Data Guard Broker*
- *Oracle9i SQL Reference*
- *Oracle9i Database Reference*
- *Oracle9i User-Managed Backup and Recovery Guide*
- *Oracle9i Recovery Manager User's Guide*
- *Oracle9i Net Services Administrator's Guide*
- *SQL*Plus User's Guide and Reference*

If you need to migrate existing standby databases to this Oracle9*i* release, see *Oracle9i Database Migration* for complete instructions. In addition, refer to *Oracle9i Database Concepts* for information about other Oracle products and features that provide disaster recovery and high data availability solutions.

In North America, printed documentation is available for sale in the Oracle Store at

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

Customers in Europe, the Middle East, and Africa (EMEA) can purchase documentation from

`http://www.oraclebookshop.com/`

Other customers can contact their Oracle representative to purchase printed documentation.

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

`http://otn.oracle.com/admin/account/membership.html`

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

```
http://otn.oracle.com/docs/index.htm
```

To access the database documentation search engine directly, please visit

```
http://tahiti.oracle.com
```

# Conventions

This section describes the conventions used in the text and code examples of this documentation set. It describes:

- Conventions in Text
- Conventions in Code Examples

## Conventions in Text

We use various conventions in text to help you more quickly identify special terms. The following table describes those conventions and provides examples of their use.

| Convention | Meaning | Example |
|---|---|---|
| **Bold** | Bold typeface indicates terms that are defined in the text or terms that appear in a glossary, or both. | When you specify this clause, you create an **index-organized table.** |
| *Italics* | Italic typeface indicates book titles or emphasis. | *Oracle9i Database Concepts*<br>Ensure that the recovery catalog and target database do *not* reside on the same disk. |
| `UPPERCASE monospace (fixed-width font)` | Uppercase monospace typeface indicates elements supplied by the system. Such elements include parameters, privileges, datatypes, RMAN keywords, SQL keywords, SQL*Plus or utility commands, packages and methods, as well as system-supplied column names, database objects and structures, usernames, and roles. | You can specify this clause only for a `NUMBER` column.<br>You can back up the database by using the `BACKUP` command.<br>Query the `TABLE_NAME` column in the `USER_TABLES` data dictionary view.<br>Use the `DBMS_STATS.GENERATE_STATS` procedure. |

| Convention | Meaning | Example |
|---|---|---|
| `lowercase monospace (fixed-width font)` | Lowercase monospace typeface indicates executables, filenames, directory names, and sample user-supplied elements. Such elements include computer and database names, network service names, and connect identifiers, as well as user-supplied database objects and structures, column names, packages and classes, usernames and roles, program units, and parameter values.<br><br>**Note:** Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown. | Enter `sqlplus` to open SQL*Plus.<br><br>The password is specified in the `orapwd` file.<br><br>Back up the datafiles and control files in the `/disk/oracle/dbs` directory.<br><br>The `department_id`, `department_name`, and `location_id` columns are in the `hr.departments` table.<br><br>Set the `QUERY_REWRITE_ENABLED` initialization parameter to `true`.<br><br>Connect as `oe` user.<br><br>The `JRepUtil` class implements these methods. |
| *`lowercase monospace (fixed-width font) italic`* | Lowercase monospace italic font represents placeholders or variables. | You can specify the *`parallel_clause`*.<br><br>Run `U`*`old_release`*`.SQL` where *`old_release`* refers to the release you installed prior to upgrading. |

## Conventions in Code Examples

Code examples illustrate SQL, PL/SQL, SQL*Plus, or other command-line statements. They are displayed in a monospace (fixed-width) font and separated from normal text as shown in this example:

```
SELECT username FROM dba_users WHERE username = 'MIGRATE';
```

The following table describes typographic conventions used in code examples and provides examples of their use.

| Convention | Meaning | Example |
|---|---|---|
| [ ] | Brackets enclose one or more optional items. Do not enter the brackets. | `DECIMAL (digits [ , precision ])` |
| { } | Braces enclose two or more items, one of which is required. Do not enter the braces. | `{ENABLE \| DISABLE}` |

| Convention | Meaning | Example |
|---|---|---|
| \| | A vertical bar represents a choice of two or more options within brackets or braces. Enter one of the options. Do not enter the vertical bar. | {ENABLE \| DISABLE}<br>[COMPRESS \| NOCOMPRESS] |
| . . . | Horizontal ellipsis points indicate either:<br>■ That we have omitted parts of the code that are not directly related to the example<br>■ That you can repeat a portion of the code | CREATE TABLE ... AS *subquery*;<br>SELECT *col1, col2, ... , coln* FROM employees; |
| .<br>.<br>. | Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example. | SQL> SELECT NAME FROM V$DATAFILE;<br>NAME<br>----------<br>/fs1/dbs/tbs_01/dbf<br>/fs1/dbs/tbs_02/dbf<br>.<br>.<br>.<br>/fs1/dbs/tbs_09/dbf<br>9 rows selected. |
| Other notation | You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as shown. | acctbal NUMBER(11,2);<br>acct CONSTANT NUMBER(4) := 3; |
| *Italics* | Italicized text indicates placeholders or variables for which you must supply particular values. | CONNECT SYSTEM/*system_password*<br>DB_NAME = *database_name* |
| UPPERCASE | Uppercase typeface indicates elements supplied by the system. We show these terms in uppercase in order to distinguish them from terms you define. Unless terms appear in brackets, enter them in the order and with the spelling shown. However, because these terms are not case sensitive, you can enter them in lowercase. | SELECT last_name, employee_id FROM employees;<br>SELECT * FROM USER_TABLES;<br>DROP TABLE hr.employees; |

| Convention | Meaning | Example |
|---|---|---|
| `lowercase` | Lowercase typeface indicates programmatic elements that you supply. For example, lowercase indicates names of tables, columns, or files.<br><br>**Note:** Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown. | `SELECT last_name, employee_id FROM employees;`<br><br>`sqlplus hr/hr`<br><br>`CREATE USER mjones IDENTIFIED BY ty3MU9;` |

# What's New in Data Guard?

This section describes the new Data Guard features in Oracle9*i* release 2 (9.2) and provides pointers to additional information. Information about new features from previous releases is retained to help those users who are upgrading to the current release.

The following sections describe the new features in Data Guard:



## Oracle9*i* Release 2 (9.2) New Features in Data Guard

The features and enhancements described in this section were added to Data Guard in Oracle9*i* release 2 (9.2).

- **Logical standby database**

  Until now, there has been only the physical standby database implementation, in which the standby database performs either managed recovery or read-only operations. A physical standby database is physically equivalent to the primary database. While redo logs are being applied to a physical standby database, it cannot be opened for reporting, and when the physical standby database is open for reporting, redo logs cannot be applied to it. A logical standby database has the same logical schema as the primary database, but it may have different physical objects, such as additional indexes. With logical standby databases, the database can be available for reporting at the same time you are applying redo logs to it.

- **Data protection modes**

  The database administrator (DBA) can place the database into one of the following modes:

  - Maximum protection
  - Maximum availability
  - Maximum performance

  These modes replace the guaranteed, instant, rapid, and delayed modes of data protection available in Oracle9*i* release 1 (9.0.1).

  **See Also:** Chapter 5, "Log Transport Services" and Chapter 13, "SQL Statements"

- **Cascading redo log destinations**

  A cascaded redo log destination is a standby database that receives its redo data from another standby database and not from the original primary database. A physical or logical standby database can be set up to send the incoming redo data to other remote destinations in the same manner as the primary database, with up to one level of redirection.

  **See Also:** Appendix D, "Cascaded Redo Log Destinations"

- **Oracle9*i* Data Guard broker**

  The broker now supports:

  - Up to nine physical or logical standby destinations
  - Failover and switchover operations

  **See Also:** *Oracle9i Data Guard Broker*

- **New keywords for the `REMOTE_ARCHIVE_ENABLE` initialization parameter include:**

  - `send`
  - `receive`

  **See Also:** Section 5.3.2.1, "Setting Permission to Archive Redo Logs"

- **New attributes for the LOG_ARCHIVE_DEST_*n* initialization parameter include:**
  - [NO]TEMPLATE

    Defines a directory specification and format for archived redo logs at the standby destination.
  - [NO]NET_TIMEOUT

    Specifies the number of seconds the log writer process will wait for status from the network server of a network operation issued by the log writer process.
  - PARALLEL qualifier to the SYNC attribute

    Indicates if I/O operations to multiple destinations are done in parallel or in series.

    **See Also:** Chapter 12, "LOG_ARCHIVE_DEST_n Parameter Attributes"

- **New syntax added to the ALTER DATABASE statement includes:**
  - ACTIVATE [PHYSICAL | LOGICAL] STANDBY DATABASE [SKIP [STANDBY LOGFILE]]
  - COMMIT TO SWITCHOVER TO {PHYSICAL | LOGICAL} {PRIMARY | STANDBY} [[WITH | WITHOUT] SESSION SHUTDOWN [WAIT | NOWAIT]]
  - [NO]FORCE LOGGING
  - RECOVER MANAGED STANDBY DATABASE [FINISH [SKIP [STANDBY LOGFILE] [WAIT | NOWAIT]]
  - RECOVER MANAGED STANDBY DATABASE [THROUGH {ALL | NEXT | LAST} SWITCHOVER]
  - RECOVER MANAGED STANDBY DATABASE [THROUGH ALL ARCHIVELOG | [ THREAD *n* ] SEQUENCE *n* ]
  - REGISTER [OR REPLACE] [PHYSICAL | LOGICAL] LOGFILE *filespec*
  - SET STANDBY DATABASE TO MAXIMIZE {PROTECTION | AVAILABILITY | PERFORMANCE}
  - START LOGICAL STANDBY APPLY

- {STOP | ABORT} LOGICAL STANDBY APPLY

  **See Also:** Chapter 13, "SQL Statements"

- **New views were added:**
  - DBA_LOGSTDBY_EVENTS
  - DBA_LOGSTDBY_LOG
  - DBA_LOGSTDBY_NOT_UNIQUE
  - DBA_LOGSTDBY_PARAMETERS
  - DBA_LOGSTDBY_PROGRESS
  - DBA_LOGSTDBY_SKIP
  - DBA_LOGSTDBY_SKIP_TRANSACTION
  - DBA_LOGSTDBY_UNSUPPORTED
  - V$DATAGUARD_STATUS
  - V$LOGSTDBY
  - V$LOGSTDBY_STATS

  **See Also:** Chapter 14, "Views"

- **New columns were added to existing fixed views:**
  - V$ARCHIVE_DEST view:
    - NET_TIMEOUT
    - TYPE
  - V$ARCHIVE_DEST_STATUS view:
    - PROTECTION_MODE
    - SRL
  - V$DATABASE view:
    - GUARD_STATUS
    - SUPPLEMENTAL_LOG_DATA_MIN
    - SUPPLEMENTAL_LOG_DATA_PK

        * SUPPLEMENTAL_LOG_DATA_UI
        * FORCE_LOGGING
        * PROTECTION_LEVEL

        **See Also:** Chapter 14, "Views"

- **Existing columns were renamed in existing fixed views:**
  - V$ARCHIVE_DEST view:
    * MANIFEST has been renamed to REGISTER and values have been changed to YES and NO.
    * REGISTER has been renamed to REMOTE_TEMPLATE.
  - V$DATABASE view:
    * STANDBY_MODE has been renamed to PROTECTION_MODE and values MAXIMUM PROTECTED, MAXIMUM AVAILABILITY, RESYNCHRONIZATION, MAXIMUM PERFORMANCE, and UNPROTECTED have been added.

    **See Also:** Chapter 14, "Views"

- **New values were added to existing columns of existing fixed views:**
  - TRANSMIT_MODE column of the V$ARCHIVE_DEST view:
    * PARALLELSYNC
    * SYNCHRONOUS
    * ASYNCHRONOUS
  - REMOTE_ARCHIVE column of the V$DATABASE view:
    * send
    * receive

    **See Also:** Chapter 14, "Views"

- **New integer values were added for the LOG_ARCHIVE_TRACE parameter:**
  - 1024: RFS physical client tracking
  - 2048: ARC*n* or RFS heartbeat tracking

**See Also:** Section 6.7.3

## Oracle9*i* Release 1 (9.0.1) New Features in Data Guard

The features and enhancements described in this section were added to Data Guard in Oracle9*i* release 1 (9.0.1).

- **Oracle9*i* Data Guard.**

  Oracle8*i* Standby Database was renamed to Oracle9*i* Data Guard.
- **Oracle9*i* Data Guard broker.**
- **No data loss.**
- **Database switchover.**
- **Archive gaps are automatically detected and transmitted.**
- **Add new datafiles to the primary database without having to manually add the corresponding datafile to the standby databases.**
- **Background managed recovery mode**.
- **Parallel recovery allows faster recovery on physical standby databases.**
- **Specify up to 10 archive destinations.**
- **Incrementally modify individual attributes of the `LOG_ARCHIVE_DEST_`*n* initialization parameter.**
- **Standby redo logs.**
- **Archiver process (ARC*n*) on physical standby databases can archive standby redo logs.**
- **Archive the current redo log, archive an online redo log based on the SCN (system change number) value when the database is mounted, but not open, or archive an online redo log when a backup control file is being used. In previous releases, a current control file was required.**
- **New control options: `DELAY`, `DISCONNECT`, `EXPIRE`, `FINISH`, `NEXT`, `NODELAY`**
- **Support for Standby Databases in Real Application Clusters.**
- **New archive log repository, which is a standalone standby database.**
- **Relationship defined between an archived redo log and an archive destination.**

- **New initialization parameters: `REMOTE_ARCHIVE_ENABLE`, `FAL_CLIENT`, `FAL_SERVER`, `STANDBY_FILE_MANAGEMENT`, `ARCHIVE_LAG_TARGET`**
- **New attributes for the `LOG_ARCHIVE_DEST_`*n*` initialization parameter include:**
  - `ARCH | LGWR`
  - `[NO]AFFIRM`
  - `[NO]ALTERNATE`
  - `[NO]DELAY`
  - `[NO]DEPENDENCY`
  - `[NO]MAX_FAILURE`
  - `[NO]QUOTA_SIZE`
  - `[NO]QUOTA_USED`
  - `[NO]REGISTER | REGISTER [=`*`location_format`*`]`
  - `NOREOPEN`
  - `SYNC | ASYNC`
- **A new range of values and the `ALTERNATE` keyword were added to the `LOG_ARCHIVE_DEST_STATE_`*n* initialization parameter.**
- **One to ten destinations (compared with one to five in Oracle8*i*) must archive successfully before the log writer process (LGWR) can overwrite the online redo logs.**
- **New tracing levels (128, 256, and 512) have been added to the `LOG_ARCHIVE_TRACE` initialization parameter.**
- **New clauses have been added to the `ALTER DATABASE` statement:**
  - `ACTIVATE [PHYSICAL] STANDBY DATABASE [SKIP [STANDBY LOGFILE]]`
  - `ADD [STANDBY] LOGFILE TO [THREAD `*`integer`*`] [GROUP `*`integer`*`] `*`filespec`*
  - `ADD [STANDBY] LOGFILE MEMBER '`*`filename`*`' [REUSE] TO '`*`logfile-descriptor`*`'`
  - `COMMIT TO SWITCHOVER TO [PHYSICAL] {PRIMARY | STANDBY} [[NO]WAIT]`

- REGISTER [PHYSICAL] LOGFILE *filespec*
- SET STANDBY DATABASE {PROTECTED | UNPROTECTED}

---

**Note:** In Oracle9*i* release 2, this syntax was further revised. See the "Oracle9i Release 2 (9.2) New Features in Data Guard" section.

---

- **New keywords were added to the RECOVER MANAGED STANDBY DATABASE clause:**
  - NODELAY
  - CANCEL [IMMEDIATE] [NOWAIT]
  - [DISCONNECT [FROM SESSION]]
  - [FINISH [NOWAIT]]
  - [PARALLEL [*integer*]]
  - NEXT
  - EXPIRE
  - DELAY

---

**Note:** In Oracle9*i* release 2, this syntax was further revised. See the "Oracle9i Release 2 (9.2) New Features in Data Guard" section.

---

- **New fixed views were added:**
  - V$ARCHIVE_DEST_STATUS
  - V$ARCHIVE_GAP
  - V$MANAGED_STANDBY
  - V$STANDBY_LOG

  **See Also:** Chapter 14, "Views"

- **New columns were added to existing fixed views:**
  - V$ARCHIVE_DEST view:
    - AFFIRM

    - ALTERNATE
    - ARCHIVER
    - ASYNC_BLOCKS
    - DELAY_MINS
    - DEPENDENCY
    - FAILURE_COUNT
    - LOG_SEQUENCE
    - MANIFEST (This column name was new in Oracle9*i* release 1; it was changed to REGISTER in Oracle9*i* release 2.)
    - MAX_FAILURE
    - MOUNTID
    - PROCESS
    - QUOTA_SIZE
    - QUOTA_USED
    - REGISTER (This column name was new in Oracle9*i* release 1; it was changed to REMOTE_TEMPLATE in Oracle9*i* release 2.)
    - SCHEDULE
    - TRANSMIT_MODE
    - TYPE
    - New values, ALTERNATE and FULL, have been added to the STATUS column.
- V$ARCHIVED_LOG view:
    - APPLIED
    - BACKUP_COUNT
    - COMPLETION_TIME
    - CREATOR
    - DELETED
    - DEST_ID
    - DICTIONARY_BEGIN

    - DICTIONARY_END
    - REGISTRAR
    - STANDBY_DEST
    - STATUS
    - END_OF_REDO
    - ARCHIVAL_THREAD#
- V$LOG view:
    - A new value, INVALIDATED, was added to the STATUS column.
- V$LOGFILE view:
    - TYPE
- V$DATABASE view:
    - ACTIVATION#
    - ARCHIVELOG_CHANGE#
    - DATABASE_ROLE
    - REMOTE_ARCHIVE
    - STANDBY_MODE
    - SWITCHOVER_STATUS
- V$ARCHIVE_DEST_STATUS view:
    - STANDBY_LOGFILE_COUNT
    - STANDBY_LOGFILE_ACTIVE

---

**Note:** In Oracle9*i* release 2, new values were added and existing columns renamed in the V$DATABASE and V$ARCHIVE_DEST fixed views. See the "Oracle9i Release 2 (9.2) New Features in Data Guard" section.

---

# Part I

## Concepts and Administration

This part contains the following chapters:

# 1

# Introduction to Oracle Data Guard

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

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



## 1.1 Data Guard Configurations

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

You can manage primary and standby databases using the command-line interface or the Data Guard broker, which includes a graphical user interface called Oracle Data Guard Manager.

### 1.1.1 Primary Database

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

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

### 1.1.2 Standby Databases

A standby database is a transactionally consistent copy of the primary database. A standby database is initially created from a backup copy of the primary database. Once created, Data Guard automatically maintains the standby database by transmitting primary database redo data to the standby system and then applying the redo logs to the standby database.

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

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

- **Physical standby database**

  Provides a physically identical copy of the primary database, with on-disk database structures that are identical to the primary database on a block-for-block basis. The database schema, including indexes, are the same. A physical standby database is kept synchronized with the primary database by recovering the redo data received from the primary database.

- **Logical standby database**

  Contains the same logical information as the production database, although the physical organization and structure of the data can be different. It is kept synchronized with the primary database by transforming the data in the redo logs received from the primary database into SQL statements and then executing the SQL statements on the standby database. A logical standby database can be used for other business purposes in addition to disaster recovery requirements. This allows users to access a logical standby database

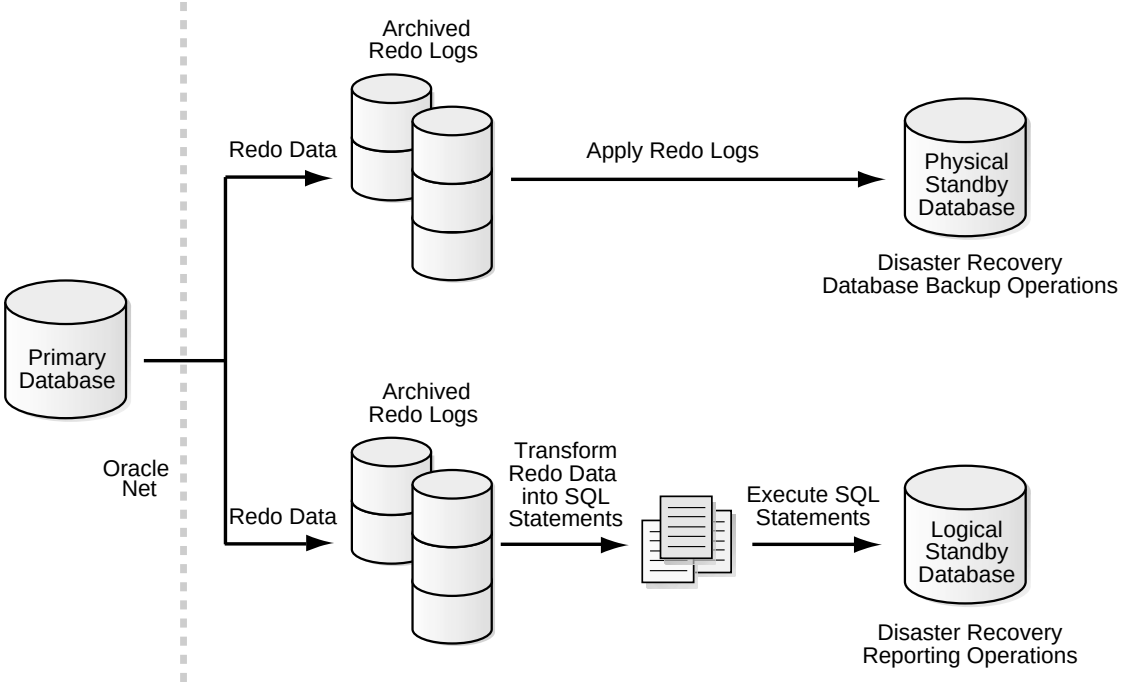
for queries and reporting purposes at any time. Thus, a logical standby database can be used concurrently for data protection and reporting.

### 1.1.3 Configuration Example

Figure 1–1 shows a Data Guard configuration that contains a primary database instance that transmits redo data to physical and logical standby databases that are both in remote locations from the primary database instance. In this configuration, a physical standby database is configured for disaster recovery and backup operations, and a logical standby database is configured primarily for reporting, but it can also be used for disaster recovery. You could configure either standby database at the same location as the primary database. However, for disaster recovery, Oracle Corporation recommends that you configure standby databases at remote locations.

Figure 1–1 shows a typical Data Guard configuration in which archived redo logs are being applied to both physical and logical standby databases.

***Figure 1–1 Typical Data Guard Configuration***



## 1.2 Data Guard Services

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

- Log Transport Services

  Control the automated transfer of redo data within a Data Guard configuration.

- Log Apply Services

  Apply archived redo logs on the standby database to maintain transactional synchronization with the primary database.

- Role Management Services

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

### 1.2.1 Log Transport Services

**Log transport services** control the automated transfer of redo data within a Data Guard configuration.

Log transport services perform the following tasks:

- Transmit redo data from the primary system to the standby systems in the configuration
- Enforce the database protection modes (described in Section 1.4)

### 1.2.2 Log Apply Services

The redo data transmitted from the primary database is archived on the standby system in the form of archived redo logs. **Log apply services** automatically apply archived redo logs on the standby database to maintain transactional synchronization with the primary database and to allow transactionally consistent read-only access to the data.

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

- For physical standby databases, Data Guard uses **redo apply** technology, which applies redo data on the standby database using standard recovery techniques of the Oracle database server.

- For logical standby databases, Data Guard uses **SQL apply** technology, which first transforms the received redo data into SQL statements and then executes the generated SQL statements on the logical standby database.

Log apply services perform the following tasks:

- Automatic application of archived redo logs on the standby database
- Automatic detection of missing redo logs on a standby system and automatic retrieval of missing redo logs from the primary database or another standby database

### 1.2.3 Role Management Services

An Oracle database operates in one of two roles: primary or standby. Using Data Guard, you can change the role of a database using either a switchover or a failover operation. The services that control these aspects are called **role management services.**

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

A **failover** is an irreversible transition of a standby database to the primary role. This is only done in the event of a catastrophic failure of the primary database. The database administrator can configure Data Guard to ensure no data loss.

## 1.3 Data Guard Broker

The Data Guard broker is a distributed management framework that automates and centralizes the creation, maintenance, and monitoring of Data Guard configurations. The following list describes some of the operations that the broker automates or simplifies:

- Creating and enabling one or more Data Guard configurations, including setting up log transport services and log apply services.
- Creating a physical or logical standby database from a backup copy of the primary database.
- Adding new or existing standby databases to an existing Data Guard configuration.

- Managing an entire Data Guard configuration from any system in the configuration.
- Monitoring log apply rates, capturing diagnostic information, and detecting problems quickly with centralized monitoring, testing, and performance tools

## 1.4 Data Guard Protection Modes

In some situations, a business cannot afford to lose data at any cost. In other situations, the availability of the database may be more important than the loss of data. Some applications require maximum database performance and can tolerate a potential loss of data.

Data Guard provides three distinct modes of data protection:

- **Maximum protection**

  This mode offers the highest level of data protection. Data is synchronously transmitted to the standby database from the primary database, and transactions are not committed on the primary database unless the redo data is available on at least one standby database configured in this mode. If the last standby database configured in this mode becomes unavailable, processing stops on the primary database. This mode guarantees no data loss.

- **Maximum availability**

  This mode is similar to the maximum protection mode, including the guarantee of no data loss. However, if a standby database becomes unavailable (for example, due to network connectivity problems), processing continues on the primary database. When the fault is corrected, the standby database is resynchronized with the primary database. If there is a need to fail over before the standby database is resynchronized, some data may be lost.

- **Maximum performance**

  This mode offers slightly less data protection on the primary database, but higher performance than maximum availability mode. In this mode, as the primary database processes transactions, redo data is asynchronously shipped to the standby database. The commit operation on the primary database does not wait for the standby database to acknowledge receipt of redo data before completing write operations on the primary database. If any standby destination becomes unavailable, processing continues on the primary database, and there is little effect on primary database performance.

## 1.5 Summary of Data Guard Benefits

Data Guard offers many overall benefits, as well as benefits provided by each kind of standby database. Note that you need to consider all benefits, including those specific to each type of standby database, when you design your Data Guard configuration.

Data Guard offers these benefits:

- Disaster recovery, data protection and high availability

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

- Complete data protection

  With its standby databases, Data Guard guarantees no data loss, even in the face of unforeseen disasters. A standby database provides a safeguard against data corruption and user errors. Storage level physical corruptions on the primary database do not propagate to the standby database. Similarly, logical corruptions or user errors that cause the primary database to be permanently damaged can be resolved. Finally, the redo data is validated when it is applied to the standby database.

- Efficient use of system resources

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

- Flexibility in data protection to balance availability against performance requirements

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

- Centralized and simple management

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

- Automatic gap detection and resolution

  If connectivity is lost between the primary and one or more standby databases (for example, due to network problems), redo data being generated on the primary database cannot be sent to those standby databases. Once connectivity is re-established, the missing log sequence (or the gap) is automatically detected by Data Guard, and the necessary redo logs are automatically transmitted to the standby databases. The standby databases are resynchronized with the primary database, with no manual intervention by the DBA.

# 2

# Getting Started with Data Guard

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

- Choosing a Standby Database Type
- Choosing a Data Guard User Interface
- Data Guard Operational Prerequisites
- Standby Database Directory Structure Considerations

## 2.1 Choosing a Standby Database Type

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

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

### 2.1.1 Physical Standby Databases

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

Data Guard maintains a physical standby database by performing managed recovery operations. When it is not performing recovery operations, a physical standby database can be open for read-only operations.

- **Managed recovery**

  The physical standby database is maintained by applying the archived redo logs on the standby system using the Oracle recovery mechanism. The recovery operation applies changes block-for-block using the physical row ID. The database cannot be opened for read or read / write operations while redo data is being applied.

- **Open read-only**

  The physical standby database can be open for read-only operations so that you can execute queries on the database. While open for read-only operations, the standby database can continue to receive redo logs but application of the data from the logs is deferred until the database resumes managed recovery operations.

Although the physical standby database cannot perform both managed recovery and read-only operations at the same time, you can switch between them. For example, you can run a physical standby database to perform managed recovery operations, then open it so applications can perform read-only operations to run reports, and then change it back to perform managed recovery operations to apply outstanding archived redo logs. You can repeat this cycle, alternating between managed recovery and read-only operations, as necessary.

In either case, the physical standby database is available to perform backup operations. Furthermore, the physical standby database will continue to receive redo logs even if they are not being applied at that moment.

#### Benefits of a Physical Standby Database

A physical standby database provides the following benefits:

- Disaster recovery and high availability

  A physical standby database enables a robust and efficient disaster recovery and high availability solution. Easy-to-manage switchover and failover capabilities allow easy role reversals between primary and physical standby

databases, minimizing the downtime of the primary database for planned and unplanned outages.

- Data protection

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

- Reduction in primary database workload

  Oracle Recovery Manager (RMAN) can use physical standby databases to off-load backups from the primary database saving valuable CPU and I/O cycles. The physical standby database can also be opened in read-only mode to perform reporting and queries.

- Performance

  The redo apply technology used by the physical standby database applies changes using low-level recovery mechanisms, which bypass all SQL level code layers and therefore is the most efficient mechanism for applying changes. This makes the redo apply technology a highly efficient mechanism to propagate changes among databases.

### 2.1.2 Logical Standby Databases

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

Data Guard automatically applies archived redo log information to the logical standby database by transforming data in the redo logs 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 open for read/write operations, its target tables for the regenerated SQL are available only for read-only operations. While those tables are being updated, they can be used simultaneously for other tasks

such as reporting, summations, and queries. Moreover, these tasks can be optimized by creating additional indexes and materialized views on the maintained tables.

A logical standby database has some restrictions on datatypes, types of tables, and types of data definition language (DDL) and data manipulation language (DML) operations. Unsupported datatypes and tables are described in more detail in Section 4.1.4.

### Benefits of a Logical Standby Database

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

- Efficient use of standby hardware resources

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

- Reduction in primary database workload

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

## 2.2 Choosing a Data Guard User Interface

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

- Command-line interface:
  - SQL*Plus

    Several SQL*Plus statements use a `STANDBY` keyword to specify operations on a standby database. Other SQL statements do not include

standby-specific syntax, but are useful for performing operations on a standby database.

**See Also:** Chapter 13 describes the relevant statements

– Initialization parameters

Several initialization parameters are used to define the Data Guard environment.

**See Also:** Section 11.3 describes relevant initialization parameters

- Data Guard broker command-line interface

  The Data Guard broker command-line interface is an alternative to using the Oracle Data Guard Manager graphical user interface (GUI). The command-line interface is useful if you want to use the broker to manage a Data Guard configuration from batch programs or scripts.

  **See Also:** *Oracle9i Data Guard Broker*

- Oracle Data Guard Manager

  The Oracle Data Guard Manager is the GUI that automates many of the tasks involved in creating, configuring, and monitoring a Data Guard environment.

  **See Also:** *Oracle9i Data Guard Broker* and the Oracle9*i* Data Guard Manager online help for information on the Data Guard Manager GUI and the Oracle9*i* Data Guard Manager Wizard

The discussions and examples in this manual use the Data Guard broker command-line interface.

## 2.3 Data Guard Operational Prerequisites

The following are operational requirements for using Data Guard:

- The same edition of Oracle Enterprise Edition must be installed on all systems in a Data Guard configuration.
- The primary database must run in ARCHIVELOG mode.
- The same Oracle software release must be used on both the primary and standby databases. The operating system running on the primary and standby

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

- The hardware and operating system architecture on the primary and standby locations must be the same. For example, this means a Data Guard configuration with a primary database on a 32-bit Sun system must have a standby database that is configured on a 32-bit Sun system. Similarly, a primary database on a 64-bit HP-UX system must be configured with a standby database on a 64-bit HP-UX system, and a primary database on a 32-bit Linux on Intel system must be configured with a standby database on a 32-bit Linux on Intel system, and so forth.
- The primary database can be a single instance database or a multi-instance Real Application Clusters database. The standby databases can be single instance databases or multi-instance Real Application Clusters databases, and these standby databases can be a mix of both physical and logical types.
- The hardware (for example, the number of CPUs, memory size, storage configuration) can be different between the primary and standby systems. If the standby system is smaller than the primary system, you may have to restrict the work that can be done on the standby system after a switchover or failover operation. Also, the standby system must have enough resources available to receive and apply all redo data from the primary database. The logical standby database requires additional resources to translate the redo data to SQL statements and then execute the SQL on the logical standby database.
- Each primary database and standby database must have its own control file.
- If you place your primary and standby databases on the same system, you must adjust the initialization parameters correctly.
- To protect against unlogged direct writes in the primary database that cannot be propagated to the standby database, turn on `FORCE LOGGING` at the primary database before performing datafile backup operations for standby creation. Keep the database in `FORCE LOGGING` mode as long as the standby database is required.
- If you are currently running Oracle Data Guard on Oracle8*i* database software, see *Oracle9i Database Migration* for complete information on upgrading to Oracle Data Guard on Oracle9*i* database software.
- The user accounts you use to manage the primary and standby database instances must have `SYSDBA` system privileges.

## 2.4 Standby Database Directory Structure Considerations

The directory structure of the various standby databases is important because it determines the path names for the standby datafiles and redo logs. 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.

For standby databases, use the same path names for the standby files if possible. Otherwise, you will need to set filename conversion parameters (as shown in Table 2–1). 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 (`Standby1`).
- A standby database on a separate system that uses the same directory structure as the primary system (`Standby2`). This is the recommended method.
- A standby database on a separate system that uses a different directory structure than the primary system (`Standby3`).

***Figure 2–1 Possible Standby Configurations***

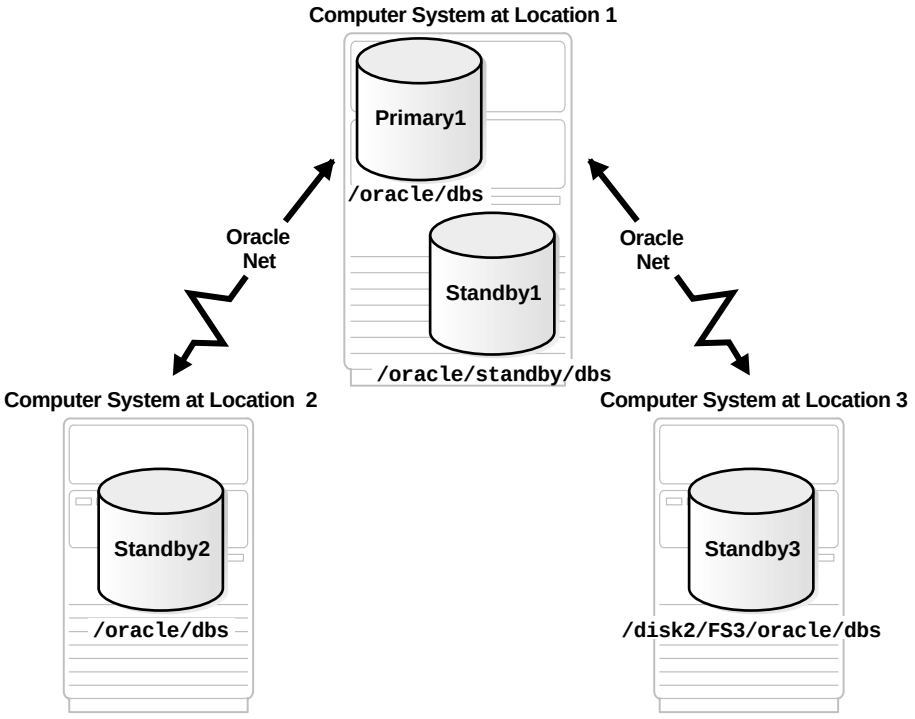



Table 2–1 describes possible configurations of primary and standby databases and the consequences of each.

***Table 2–1 Standby Database Location and Directory Options***

| Standby System | Directory Structure | Consequences |
|---|---|---|
| Same as primary system | Different than primary system (required) | ■ You must set the LOCK_NAME_SPACE initialization parameter.<br>■ You must manually rename primary database datafiles and redo logs in the standby database control file (see Appendix B). For physical standby databases, you could alternatively set up the DB_FILE_NAME_CONVERT and LOG_FILE_NAME_CONVERT initialization parameters on the standby database to automatically rename the datafiles (see Section 6.2.4).<br>■ The standby database does not protect against disasters that destroy the system on which both 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 and redo logs 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).<br>■ Using separate physical media for your databases safeguards your primary data. |
| Separate system | Different than primary system | ■ You must manually rename primary database datafiles and redo logs in the standby database control file (see Appendix B). For physical standby databases, you could alternatively set up the DB_FILE_NAME_CONVERT and LOG_FILE_NAME_CONVERT initialization parameters on the standby database to automatically rename the datafiles (see Section 6.2.4).<br>■ Using separate physical media for your databases safeguards your primary data. |

# 3

# Creating a Physical Standby Database

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

- Preparing the Primary Database for Standby Database Creation
- Creating a Physical Standby Database
- Verifying the Physical Standby Database

The discussions in this chapter assume that you specify initialization parameters in a server parameter file (SPFILE) instead of in a traditional text initialization parameter file (PFILE). See the *Oracle9i Database Administrator's Guide* for information about creating and using server parameter files.

> **See Also:** *Oracle9i Data Guard Broker* and the Oracle Data Guard Manager online help system for information about using the Data Guard Manager graphical user interface to automatically create a physical standby database

## 3.1 Preparing the Primary Database for Standby Database Creation

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

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

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

| Reference | Task |
|---|---|
| Section 3.1.1 | Enable Forced Logging |

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

| Reference | Task |
|---|---|
| Section 3.1.2 | Enable Archiving and Define a Local Archiving Destination |

### 3.1.1 Enable Forced Logging

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

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

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

### 3.1.2 Enable Archiving and Define a Local Archiving Destination

Ensure that the primary database is in ARCHIVELOG mode, that automatic archiving is enabled, and that you have defined a local archiving destination.

Set the local archive destination using the following SQL statement:

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_1='LOCATION=/disk1/oracle/oradata/payroll
  2> MANDATORY' SCOPE=BOTH;
```

> **See Also:** *Oracle9i Database Administrator's Guide* for a description of archiving and Chapter 11 and the *Oracle9i Database Reference* for information about initialization parameters

## 3.2 Creating a Physical Standby Database

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

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

*Table 3–2 Creating a Physical Standby Database*

| Reference | Task | Database |
|---|---|---|
| Section 3.2.1 | Identify the Primary Database Datafiles | Primary |
| Section 3.2.2 | Make a Copy of the Primary Database | Primary |
| Section 3.2.3 | Create a Control File for the Standby Database | Primary |
| Section 3.2.4 | Prepare the Initialization Parameter File to be Copied to the Standby Database | Primary |
| Section 3.2.5 | Copy Files from the Primary System to the Standby System | Primary |
| Section 3.2.6 | Set Initialization Parameters on a Physical Standby Database | Standby |
| Section 3.2.7 | Create a Windows Service | Standby |
| Section 3.2.8 | Configure Listeners for the Primary and Standby Databases | Primary and Standby |
| Section 3.2.9 | Enable Dead Connection Detection on the Standby System | Standby |
| Section 3.2.10 | Create Oracle Net Service Names | Primary and Standby |
| Section 3.2.11 | Create a Server Parameter File for the Standby Database | Standby |
| Section 3.2.12 | Start the Physical Standby Database | Standby |
| Section 3.2.13 | Initiate Log Apply Services | Standby |
| Section 3.2.14 | Enable Archiving to the Physical Standby Database | Primary |

## 3.2.1 Identify the Primary Database Datafiles

On the primary database, query the V$DATAFILE view to list the files that will be used to create the physical standby database, as follows:

```
SQL> SELECT NAME FROM V$DATAFILE;
NAME
--------------------------------------------------------------------------------
/disk1/oracle/oradata/payroll/system01.dbf
/disk1/oracle/oradata/payroll/undotbs01.dbf
/disk1/oracle/oradata/payroll/cwmlite01.dbf
.
.
.
```

## 3.2.2 Make a Copy of the Primary Database

On the primary database, perform the following steps to make a closed backup copy of the primary database.

**Step 1 Shut down the primary database.**

Issue the following SQL*Plus statement to shut down the primary database:

```
SQL> SHUTDOWN IMMEDIATE;
```

**Step 2 Copy the datafiles to a temporary location.**

Copy the datafiles that you identified in Section 3.2.1 to a temporary location using an operating system utility copy command. The following example uses the UNIX `cp` command:

```
cp /disk1/oracle/oradata/payroll/system01.dbf
/disk1/oracle/oradata/payroll/standby/system01.dbf
```

Copying the datafiles to a temporary location will reduce the amount of time that the primary database must remain shut down.

**Step 3 Restart the primary database.**

Issue the following SQL*Plus statement to restart the primary database:

```
SQL> STARTUP;
```

## 3.2.3 Create a Control File for the Standby Database

On the primary database, create the control file for the standby database, as shown in the following example:

```
SQL> ALTER DATABASE CREATE STANDBY CONTROLFILE AS
  2> '/disk1/oracle/oradata/payroll/standby/payroll2.ctl';
```

---

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

---

The filename for the newly created standby control file *must* be different from the filename of the current control file of the primary database. The control file must also be created after the last time stamp for the backup datafiles.

### 3.2.4 Prepare the Initialization Parameter File to be Copied to the Standby Database

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

```
SQL> CREATE PFILE='/disk1/oracle/dbs/initpayroll2.ora' FROM SPFILE;
```

Later, in Section 3.2.11, you will convert this file back to a server parameter file after it is modified to contain the parameter values appropriate for use with the physical standby database.

### 3.2.5 Copy Files from the Primary System to the Standby System

On the primary system, use an operating system copy utility to copy the following binary files from the primary system to the standby system:

- Backup datafiles created in Section 3.2.2
- Standby control file created in Section 3.2.3
- Initialization parameter file created in Section 3.2.4

### 3.2.6 Set Initialization Parameters on a Physical Standby Database

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

Example 3–1 shows the portion of the standby initialization parameter file where values were modified for the physical standby database. Parameter values that changed are shown in bold typeface.

***Example 3–1 Modifying Initialization Parameters for a Physical Standby Database***

```
.
.
.
db_name=PAYROLL
compatible=9.2.0.1.0
control_files='/disk1/oracle/oradata/payroll/standby/payroll2.ctl'
log_archive_start=TRUE
standby_archive_dest='/disk1/oracle/oradata/payroll/standby'
db_file_name_convert=('/disk1/oracle/oradata/payroll/',
'/disk1/oracle/oradata/payroll/standby/')
log_file_name_convert=('/disk1/oracle/oradata/payroll/',
```

```
'/disk1/oracle/oradata/payroll/standby/')
log_archive_format=log%d_%t_%s.arc
log_archive_dest_1=('LOCATION=/disk1/oracle/oradata/payroll/standby/')
standby_file_management=AUTO
remote_archive_enable=TRUE
instance_name=PAYROLL2
# The following parameter is required only if the primary and standby databases
# are located on the same system.
lock_name_space=PAYROLL2
.
.
.
```

The following list provides a brief explanation about the parameter settings shown in Example 3–1:

- `db_name` - Not modified. The same name as the primary database.
- `compatible` - Not modified. The same as the primary database, 9.2.0.1.0.
- `control_files` - Specify the path name and filename for the standby control file.
- `log_archive_start` - Not modified. The same as the setting for the primary database, `TRUE`.
- `standby_archive_dest` - Specify the location of the archived redo logs that will be received from the primary database.
- `db_file_name_convert` - Specify the location of the primary database datafiles followed by the standby location of the datafiles. This parameter will convert the filename of the primary database datafiles to the filename of the standby datafile filenames. If the standby database is on the same system as the primary database or if the directory structure where the datafiles are located on the standby site is different from the primary site then this parameter is required. See Section 3.2.1 for the location of the datafiles on the primary database.
- `log_file_name_convert` - Specify the location of the primary database logs followed by the standby location of the logs. This parameter will convert the filename of the primary database log to the filenames of the standby log. If the standby database is on the same system as the primary database or if the directory structure where the logs are located on the standby site is different from the primary site then this parameter is required. See Section 3.2.1 for the location of the logs on the primary database.

- log_archive_format - Specify the format for the archived redo logs using a DBID (%d), thread (%t), and sequence number (%s).
- log_archive_dest_1 - Specify the location where the redo logs are to be archived on the standby system. (If a switchover occurs and this instance becomes the primary database, then this parameter will specify the location where the online redo logs will be archived.)
- standby_file_management - Set to AUTO.
- remote_archive_enable - Set to TRUE.
- instance_name - If this parameter is defined, specify a different value for the standby database than the primary database when the primary and standby databases reside on the same host.
- lock_name_space - Specify the standby database instance name.

  Use this parameter when you create the physical standby database on the same system as the primary database. Change the INSTANCE_NAME parameter to a value other than its primary database value, and set this LOCK_NAME_SPACE initialization parameter to the same value that you specified for the standby database INSTANCE_NAME initialization parameter.

---

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

---

**See Also:** Chapter 11 for a complete explanations of all the initialization parameters that can be used to modify a Data Guard environment

### 3.2.7 Create a Windows Service

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

```
WINNT> oradim -NEW -SID payroll2 -STARTMODE manual
```

**See Also:** *Oracle9i Database Administrator's Guide for Windows* for more information about using the ORADIM utility

### 3.2.8 Configure Listeners for the Primary and Standby Databases

On both the primary and standby sites, use Oracle Net Manager to configure a listener for the respective databases. If you plan to manage the configuration using the Data Guard broker, you must configure the listener to use the TCP/IP protocol and statically register service information for each database using the SID for the database instance.

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

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

**See Also:** *Oracle9i Net Services Administrator's Guide*

### 3.2.9 Enable Dead Connection Detection on the Standby System

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

```
SQLNET.EXPIRE_TIME=2
```

### 3.2.10 Create Oracle Net Service Names

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

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

**See Also:** *Oracle9i Net Services Administrator's Guide* and the *Oracle9i Database Administrator's Guide*

### 3.2.11 Create a Server Parameter File for the Standby Database

On an idle standby database, use the SQL CREATE statement to create a server parameter file for the standby database from the text initialization parameter file that was edited in Section 3.2.6. For example:

```
SQL> CREATE SPFILE FROM PFILE='initpayroll2.ora';
```

### 3.2.12 Start the Physical Standby Database

On the standby database, issue the following SQL statements to start and mount the database in standby mode:

```
SQL> STARTUP NOMOUNT;
SQL> ALTER DATABASE MOUNT STANDBY DATABASE;
```

### 3.2.13 Initiate Log Apply Services

On the standby database, start log apply services as shown in the following example:

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

The example includes the DISCONNECT FROM SESSION option so that log apply services run in a background session.

> **See Also:** Section 6.2, "Applying Redo Data to Physical Standby Databases"

### 3.2.14 Enable Archiving to the Physical Standby Database

This section describes the minimum amount of work you must do on the primary database to set up and enable archiving to the physical standby database.

> **See Also:** Chapter 5 for information about log transport services and Chapter 12 for reference information about additional attributes you can set on the LOG_ARCHIVE_DEST_*n* initialization parameter

**Step 1 Set initialization parameters to define archiving.**

To configure archive logging from the primary database to the standby site the LOG_ARCHIVE_DEST_*n* and LOG_ARCHIVE_DEST_STATE_*n* parameters must be defined.

The following example sets the initialization parameters needed to enable archive logging to the standby site:

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_2='SERVICE=payroll2' SCOPE=BOTH;
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2=ENABLE SCOPE=BOTH;
```

**Step 2 Start remote archiving.**

Archiving of redo logs to the remote standby location does not occur until after a log switch. A log switch occurs, by default, when an online redo log becomes full. To force the current redo logs to be archived immediately, use the SQL ALTER SYSTEM statement on the primary database. For example:

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

> **See Also:** Section 6.2, "Applying Redo Data to Physical Standby Databases"

## 3.3 Verifying the Physical Standby Database

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

To see the new archived redo logs that were received on the standby database, you should first identify the existing archived redo logs on the standby database, archive a few logs on the primary database, and then check the standby database again. The following steps show how to perform these tasks.

**Step 1 Identify the existing archived redo logs.**

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

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

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

3 rows selected.
```

### Step 2 Archiving the current log.

On the primary database, archive the current log using the following SQL statement:

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

### Step 3 Verify that the new archived redo log was received.

On the standby database, query the V$ARCHIVED_LOG view to verify the redo log was received:

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

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

4 rows selected.
```

The logs are now available for log apply services to apply redo data to the standby database.

### Step 4 Verify that the new archived redo log was applied.

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

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

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

4 rows selected.
```

**See Also:** Section 5.9, "Monitoring Redo Log Archival Information" and Section 6.5, "Monitoring Log Apply Services for Physical Standby Databases" for information about how to verify that both log transport services and log apply services are working correctly

# 4

# Creating a Logical Standby Database

This chapter steps you through the process of creating a logical standby database and then configuring log apply services to maintain the standby database using SQL apply technology. It includes the following main topics:

- Preparing the Primary Database for Standby Database Creation
- Creating a Logical Standby Database
- Verify the Logical Standby Database

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

> **See Also:** *Oracle9i Data Guard Broker* and the Oracle Data Guard Manager online help system for information about using the Data Guard Manager graphical user interface to automatically create a logical standby database

## 4.1 Preparing the Primary Database for Standby Database Creation

Before performing the tasks described in this chapter, you must ensure that the user account you use on the primary database during the logical standby database creation process is configured to have the following database roles:

- `LOGSTDBY_ADMINISTRATOR` role (to use the logical standby functionality)
- `SELECT_CATALOG_ROLE` role (to have `SELECT` privileges on all data dictionary views)

Also, the discussions in this chapter assume that you specify initialization parameters in a server parameter file (SPFILE), instead of a text initialization parameter file (PFILE).

**See Also:** *Oracle9i Database Administrator's Guide* for information about creating and using server parameter files

Table 4–1 provides a checklist of the tasks that you perform on the primary database to prepare for logical standby database creation. There is also a reference to the section that describes the task in more detail.

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

| Reference | Task |
|---|---|
| Section 4.1.1 | Enable Forced Logging |
| Section 4.1.2 | Enable Archiving and Define a Local Archiving Destination |
| Section 4.1.3 | Verify the LOG_PARALLELISM Initialization Parameter |
| Section 4.1.4 | Determine Support for Datatypes or Tables |
| Section 4.1.5 | Ensure That Table Rows in the Primary Database Can Be Uniquely Identified |
| Section 4.1.6 | Ensure That Supplemental Logging Is Enabled |
| Section 4.1.7 | Create an Alternate Tablespace |

**Note:** Perform the steps listed in Table 4–1 only once. After you complete these steps, the database is prepared to serve as the primary database for one or more logical standby databases.

## 4.1.1 Enable Forced Logging

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

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

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

## 4.1.2 Enable Archiving and Define a Local Archiving Destination

Ensure that the primary database is in ARCHIVELOG mode, that automatic archiving is enabled, and that you defined a local archiving destination.

Set the local archive destination using the following SQL statement:

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_1='LOCATION=/disk1/oracle/oradata/payroll
  2> MANDATORY' SCOPE=BOTH;
```

> **See Also:** *Oracle9i Database Administrator's Guide* for a description of archiving, and Chapter 11 and the *Oracle9i Database Reference* for information about initialization parameters

### 4.1.3 Verify the LOG_PARALLELISM Initialization Parameter

On the primary database, use the SHOW PARAMETER *parameter_name* statement to determine the current values of the LOG_PARALLELISM initialization parameter. Logical standby databases require that you set this initialization parameter to one, which is the default value. If the LOG_PARALLELISM initialization parameter is already set to one, then skip to Section 4.1.4. Otherwise, set LOG_PARALLELISM=1 by issuing the SQL ALTER SYSTEM SET statement on the primary database and include the SCOPE=SPFILE clause to ensure the value is updated in the server parameter file. For example:

```
SQL> ALTER SYSTEM SET LOG_PARALLELISM=1 SCOPE=SPFILE;
```

If you change the LOG_PARALLELISM initialization parameter, you must shut down and restart the primary database so that the new initialization parameter value will take effect. For example:

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

> **See Also:** *Oracle9i SQL Reference* for more information about the ALTER SET statement, and Chapter 11 and the *Oracle9i Database Reference* for information about initialization parameters

### 4.1.4 Determine Support for Datatypes or Tables

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

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

#### Supported Datatypes

CHAR
NCHAR
VARCHAR2 and VARCHAR

NVARCHAR2
NUMBER
DATE
TIMESTAMP
TIMESTAMP WITH TIME ZONE
TIMESTAMP WITH LOCAL TIME ZONE
INTERVAL YEAR TO MONTH
INTERVAL DAY TO SECOND
RAW
CLOB
BLOB

### Unsupported Datatypes

NCLOB
LONG
LONG RAW
BFILE
ROWID
UROWID
user-defined types
object types REFs
varrays
nested tables

### Unsupported Tables, Sequences, and Views

User-defined tables and sequences in the SYS schema
Tables with unsupported datatypes
Tables using data segment compression
Index-organized tables

To determine if the primary database contains unsupported objects, query the DBA_LOGSTDBY_UNSUPPORTED view. For example, use the following query on the primary database to list the schema and table names of primary database tables that are not supported by logical standby databases:

```
SQL> SELECT DISTINCT OWNER,TABLE_NAME FROM DBA_LOGSTDBY_UNSUPPORTED
  2> ORDER BY OWNER,TABLE_NAME;

OWNER        TABLE_NAME
-----------  --------------------------
HR           COUNTRIES
OE           ORDERS
```

```
OE            CUSTOMERS
OE            WAREHOUSES
.
.
.
```

To view the column names and datatypes for one of the tables listed in the previous query, use a SELECT statement similar to the following:

```
SQL> SELECT COLUMN_NAME,DATA_TYPE FROM DBA_LOGSTDBY_UNSUPPORTED
  2> WHERE OWNER='OE' AND TABLE_NAME = 'CUSTOMERS';

COLUMN_NAME                    DATA_TYPE
------------------------------ ------------------
CUST_ADDRESS                   CUST_ADDRESS_TYP
PHONE_NUMBERS                  PHONE_LIST_TYP
CUST_GEO_LOCATION              SDO_GEOMETRY
```

If the primary database contains unsupported tables, log apply services automatically exclude these tables when applying redo logs to the logical standby database.

---

**Note:** If you determine that the critical tables in your primary database are not supported by logical standby databases, then you might want to consider using a physical standby database. See Chapter 3 for information about creating a physical standby database.

---

**See Also:** Chapter 14, "Views" for more information about the DBA_LOGSTDBY_UNSUPPORTED view

### Skipped SQL Statements on a Logical Standby Database

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

```
ALTER DATABASE
ALTER SESSION
ALTER SNAPSHOT
ALTER SNAPSHOT LOG
ALTER SYSTEM SWITCH LOG
CREATE CONTROL FILE
CREATE DATABASE
```

```
CREATE DATABASE LINK
CREATE PFILE FROM SPFILE
CREATE SCHEMA AUTHORIZATION
CREATE SNAPSHOT
CREATE SNAPSHOT LOG
CREATE SPFILE FROM PFILE
CREATE TABLE AS SELECT FROM A CLUSTER TABLE
DROP DATABASE LINK
DROP SNAPSHOT
DROP SNAPSHOT LOG
EXPLAIN
LOCK TABLE
RENAME
SET CONSTRAINTS
SET ROLE
SET TRANSACTION
```

#### Determine Support for Objects and Operations

PL/SQL procedures that modify metadata are not applied on the standby database, therefore their effects are not visible on the standby database. An example of this is the DBMS_AQADM advanced queuing package, which is not supported by logical standby databases. Another example is DBMS_MVIEW_REFRESH, which when executed on the primary database, is not maintained by SQL apply operations on the standby database.

---

**Note:** The DBMS_MVIEW_REFRESH routine must be invoked on the standby database where you want to refresh the materialized views.

---

The only exception to this is the DBMS_JOB package. Jobs metadata is applied to the logical standby database, but jobs are not executed.

### 4.1.5 Ensure That Table Rows in the Primary Database Can Be Uniquely Identified

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

- Primary key

- Unique index

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

Perform the actions described in Section 4.1.5.1 and Section 4.1.5.2 to ensure that log apply services can uniquely identify table rows.

#### 4.1.5.1 Finding Tables Without a Unique Identifier in the Primary Database

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

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

Some of the tables displayed in the DBA_LOGSTDBY_NOT_UNIQUE view can still be supported because **supplemental logging** (that you will enable in Section 4.1.6) adds information that uniquely identifies the row in the redo logs. The presence or absence of a primary key or unique index can affect supplemental logging as follows:

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

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

- Y

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

- N

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

#### 4.1.5.2 Adding a Disabled Primary Key RELY Constraint

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

> **See also:** *Oracle9i SQL Reference* for ALTER TABLE statement syntax and usage information

To create a disabled RELY constraint on a primary database table:

Use the ALTER TABLE statement with a RELY DISABLE clause. The following example creates a disabled RELY constraint on a table named mytab where rows can be uniquely identified using the id and name columns:

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

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

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

> **See Also:** Chapter 14, "Views" for more information about the DBA_LOGSTDBY_NOT_UNIQUE view and *Oracle9i SQL Reference* for more information about creating RELY constraints, and Section 9.2 for information about RELY constraints and actions you can take to increase performance on a logical standby database

### 4.1.6 Ensure That Supplemental Logging Is Enabled

Supplemental logging must be enabled on the primary database before you create the logical standby database. Because Oracle only logs the columns that were modified, this is not always sufficient to uniquely identify the row that changed and additional (supplemental) information must be put into the redo log. The supplemental information that is added to the redo logs helps log apply services to correctly identify and maintain tables in the logical standby database.

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

```
SQL> SELECT SUPPLEMENTAL_LOG_DATA_PK, SUPPLEMENTAL_LOG_DATA_UI FROM V$DATABASE;
SUP SUP
--- ---
NO NO
```

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

If supplemental logging is enabled, then go to Section 4.1.7. If supplemental logging is not enabled, then perform the steps in the following sections to enable supplemental logging.

#### 4.1.6.1 Enable Supplemental Logging

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

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

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

#### 4.1.6.2 Switch to a New Redo Log

On the primary database, issue the following statement to switch to a new redo log:

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

By switching to a new log file, you ensure that the redo logs do not contain both supplemental log data and nonsupplemental log data. Logical standby databases cannot use a redo log that contains both supplemental and nonsupplemental log data.

#### 4.1.6.3 Verify That Supplemental Logging Is Enabled

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

```
SQL> SELECT SUPPLEMENTAL_LOG_DATA_PK, SUPPLEMENTAL_LOG_DATA_UI FROM V$DATABASE;
SUP SUP
--- ---
YES YES
```

In this example, the YES values indicate that supplemental logging is enabled on the primary database. For all tables with a primary key (SUPPLEMENTAL_LOG_DATA_PK) or unique index (SUPPLEMENTAL_LOG_DATA_UI), all columns of the

primary key and unique index are placed into the redo log whenever an update operation is performed.

---

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

---

**See Also:** Chapter 14, "Views" for more information about the V$DATABASE view and the *Oracle9i SQL Reference* for more information about the ALTER DATABASE ADD SUPPLEMENTAL LOG DATA statements

### 4.1.7 Create an Alternate Tablespace

If you expect to perform switchover operations between the primary database and a logical standby database, you should create an alternate tablespace in the primary database and move the logical standby system tables to that separate tablespace.

Logical standby databases use a number of tables defined in the SYS and SYSTEM schemas. By default, these tables are created in the SYSTEM tablespace. Some of these tables can rapidly become very large. By preparing an alternate tablespace in advance and moving the logical standby system tables to a separate tablespace, you will prevent these tables from filling the entire SYSTEM tablespace. Move the tables to the new tablespace before they are populated during the logical standby creation process described in Section 4.2.

To create a new tablespace for the logical standby tables, issue the SQL CREATE TABLESPACE statement. Then, use the DBMS_LOGMNR_D.SET_TABLESPACE procedure to move the tables into the new tablespace on the primary database. For example, the following statements create a new tablespace named logmnrts and move the LogMiner tables into that tablespace:

```
SQL> CREATE TABLESPACE logmnrts DATAFILE '/disk1/oracle/dbs/logmnrts.dbf'
  2> SIZE 25M AUTOEXTEND ON MAXSIZE UNLIMITED;
SQL> EXECUTE DBMS_LOGMNR_D.SET_TABLESPACE('logmnrts');
```

Note that creating an alternate tablespace can take several minutes to complete.

By creating an alternate tablespace on the primary database, any standby databases created from the primary database will contain the new tablespace. If the primary database later becomes a standby database, it will be set up correctly.

---

**Note:** If this primary database is part of a Data Guard configuration that already contains a standby database and the standby database has the initialization parameter STANDBY_FILE_MANAGEMENT set to AUTO, then the previous commands will automatically be applied to the standby database. If the STANDBY_FILE_MANAGEMENT initialization parameter is *not* set to AUTO, then the previous commands must be issued on the standby database to ensure that future switchovers work correctly.

---

**See Also:** *Oracle9i SQL Reference* for information about the CREATE TABLESPACE statement and the *Oracle9i Supplied PL/SQL Packages and Types Reference* for information about the DBMS_LOGMNR_D supplied package

## 4.2 Creating a Logical Standby Database

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

***Table 4–2 Create a Logical Standby Database***

| Reference | Task | Database |
|---|---|---|
| Section 4.2.1 | Identify the Primary Database Datafiles and Log Files | Primary |
| Section 4.2.2 | Make a Copy of the Primary Database | Primary |
| Section 4.2.3 | Prepare the Initialization Parameter File to Be Copied to the Standby System | Primary |
| Section 4.2.4 | Copy Files from the Primary Database Location to the Standby Location | Primary |
| Section 4.2.5 | Set Initialization Parameters on the Logical Standby Database | Standby |
| Section 4.2.6 | Create a Windows Service | Standby |
| Section 4.2.7 | Configure the Listener for Both the Primary and Standby Databases | Primary and Standby |
| Section 4.2.8 | Enable Dead Connection Detection on the Standby System | Standby |

*Table 4–2 Create a Logical Standby Database*

| Reference | Task | Database |
|---|---|---|
| Section 4.2.9 | Create Oracle Net Service Names | Primary and Standby |
| Section 4.2.10 | Start and Mount the Logical Standby Database | Standby |
| Section 4.2.11 | Rename the Datafiles on the Logical Standby Database | Standby |
| Section 4.2.12 | Rename Online Redo Logs on the Logical Standby Database | Standby |
| Section 4.2.13 | Turn On the Database Guard | Standby |
| Section 4.2.14 | Reset the Database Name of the Logical Standby Database | Standby |
| Section 4.2.15 | Change the Database Name in the Parameter File | Standby |
| Section 4.2.16 | Create a New Temporary File for the Logical Standby Database | Standby |
| Section 4.2.17 | Register the Archived Redo Log and Start SQL Apply Operations | Standby |
| Section 4.2.18 | Enable Archiving to the Logical Standby Database | Primary |

**Note:** Perform the steps listed in Table 4–2 for each logical standby database that you want to create.

## 4.2.1 Identify the Primary Database Datafiles and Log Files

On the primary database, query the V$DATAFILE view to list the files that will be used to create the logical standby database. For example:

```
SQL> SELECT NAME FROM V$DATAFILE;
NAME
--------------------------------------------------------------------------------
/disk1/oracle/oradata/payroll/system01.dbf
/disk1/oracle/oradata/payroll/undotbs01.dbf
/disk1/oracle/oradata/payroll/cwmlite01.dbf
.
.
.
```

On the primary database, query the V$LOGFILE view to list the primary database logs. (This information will be used in later steps.) For example:

```
SQL> SELECT GROUP#,TYPE,MEMBER FROM V$LOGFILE;
GROUP#            TYPE       MEMBER
```

```
-------------------------------------------------------------------------------
1          ONLINE          /disk1/oracle/oradata/payroll/redo01.log
2          ONLINE          /disk1/oracle/oradata/payroll/redo02.log
3          ONLINE          /disk1/oracle/oradata/payroll/redo03.log
.
.
.
```

## 4.2.2 Make a Copy of the Primary Database

Perform the following steps to make a closed backup copy of the primary database.

### Step 1 Shut down the primary database.

On the primary database, use the following SQL*Plus statement to shut it down:

```
SQL> SHUTDOWN IMMEDIATE;
```

### Step 2 Copy the datafiles to a temporary location.

On the primary database, copy the datafiles that you identified in Section 4.2.1 to a temporary location using an operating system utility copy command. The following example uses the UNIX cp command:

```
cp /disk1/oracle/oradata/payroll/system01.dbf
/disk1/oracle/oradata/payroll/standby/system01.dbf

cp /disk1/oracle/oradata/payroll/undotbs01.dbf
/disk1/oracle/oradata/payroll/standby/undotbs01.dbf

cp /disk1/oracle/oradata/payroll/cwmlite01.dbf
/disk1/oracle/oradata/payroll/standby/cwmlite01.dbf
.
.
.
```

Copying the datafiles to a temporary location reduces the amount of time that the primary database must remain shut down.

### Step 3 Restart the primary database.

On the primary database, use the following SQL*Plus command to restart and mount the primary database:

```
SQL> STARTUP MOUNT;
```

### Step 4 Create a backup copy of the control file for the standby database.

On the primary database, create a backup copy of the control file for the standby database:

```
SQL> ALTER DATABASE BACKUP CONTROLFILE TO
  2> '/disk1/oracle/oradata/payroll/standby/payroll3.ctl';
```

### Step 5 Enable restricted session mode on the primary database.

On the primary database, enable restricted session mode to reduce the likelihood of users or applications performing any DML or DDL operations.

> **Caution:** Do not allow any DML or DDL operations to occur until after restricted session mode is disabled in step 7.

The following statement enables restricted session mode:

```
SQL> ALTER SYSTEM ENABLE RESTRICTED SESSION;
```

### Step 6 Build the LogMiner dictionary.

To create a logical standby database, you must manually build the dictionary for the logical standby database. On the primary database, issue the following statements to build the LogMiner dictionary:

```
SQL> ALTER DATABASE OPEN;
SQL> EXECUTE DBMS_LOGSTDBY.BUILD;
```

### Step 7 Disable restricted session mode on the primary database.

On the primary database, disable restricted session mode using the following SQL statement:

```
SQL> ALTER SYSTEM DISABLE RESTRICTED SESSION;
```

### Step 8 Identify the latest archived redo log.

To obtain a starting point for building the logical standby database, query the V$ARCHIVED_LOG view, identify the latest archived redo log, and record its name for use later in the creation process. The following query includes the DICTIONARY_BEGIN clause to find the name of the new dictionary and the STANDBY_DEST clause to show information only for the local destination. (Without the STANDBY_DEST clause, the information will include output for both the local and standby destinations.) For example:

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

```
SQL> SELECT NAME FROM V$ARCHIVED_LOG
  2> WHERE (SEQUENCE#=(SELECT MAX(SEQUENCE#) FROM V$ARCHIVED_LOG
  3> WHERE DICTIONARY_BEGIN = 'YES' AND STANDBY_DEST= 'NO'));

NAME
----------------------------------------
/disk1/oracle/oradata/payroll/arc0004.001
```

Remember to record the name of the archived redo log for use later in the creation process.

### 4.2.3 Prepare the Initialization Parameter File to Be Copied to the Standby System

Create a text initialization parameter file from the server parameter file used by the primary database; a text initialization parameter file can be copied to the standby location and modified. The following example shows the statement you use on the primary database to create a text initialization parameter file from an spfile:

```
SQL> CREATE PFILE='/disk1/oracle/dbs/initpayroll3.ora' FROM SPFILE;
```

Later, in Section 4.2.15, you will convert this file back to a server parameter file after it is modified to contain the parameter values appropriate for use with the logical standby database.

### 4.2.4 Copy Files from the Primary Database Location to the Standby Location

On the primary database, use an operating system copy utility to copy the following binary files from the primary database site to the standby site:

- Backup datafiles and control files created in Section 4.2.2.
- Latest archived redo log that was identified in step 8 of Section 4.2.2.
- Database initialization parameter file created in Section 4.2.3.

### 4.2.5 Set Initialization Parameters on the Logical Standby Database

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

Example 4–1 shows the portion of the standby text initialization parameter file where values have been modified for the logical standby database. Parameter values that were changed are shown in bold typeface.

***Example 4–1 Modifying Initialization Parameters for a Logical Standby Database***

```
.
.
.
db_name=PAYROLL
compatible=9.2.0.1.0
control_files='/disk1/oracle/oradata/payroll/standby/payroll3.ctl'
log_archive_start=TRUE
standby_archive_dest='/disk1/oracle/oradata/payroll/standby'
log_archive_format=log%d_%t_%s.arc
log_archive_dest_1='LOCATION=/disk1/oracle/oradata/payroll/arch/'
log_parallelism=1
parallel_max_servers=9
instance_name=PAYROLL3
# The following parameter is required only if the primary and standby databases
# are located on the same system.
lock_name_space=PAYROLL3
.
.
.
```

The following list provides a brief explanation about the setting of the parameters shown in Example 4–1:

- `db_name` - Not modified. The same name as the primary database.
- `compatible` - Not modified. The same as the primary database, 9.2.0.1.0.
- `control_files` - Specify the path name and filename for the standby control file.
- `log_archive_start` - Not modified. The same as the setting for the primary database, `TRUE`.
- `standby_archive_dest` - Specify the location of the archived redo logs that will be received from the primary database.
- `log_archive_format` - Specify the format for the archived redo logs using a DBID (%d), thread (%t), and sequence number (%s).
- `log_archive_dest_1` - Specify the location where the redo logs are to be archived.
- `log_parallelism` - Not modified. The same value as the primary database.
- `parallel_max_servers` - Set to 9.

- instance_name - If this parameter is defined, specify a different value for the standby database than the primary database when the primary and standby databases reside on the same host.
- lock_name_space - Specify the standby database instance name.

  Use this parameter when you create the logical standby database on the same system as the primary database. You must change the INSTANCE_NAME parameter to a value other than its primary database value, and set this LOCK_NAME_SPACE initialization parameter to the same value that you specified for the standby database INSTANCE_NAME initialization parameter.

---

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

---

**See Also:** Chapter 11 for a complete explanations of all the initialization parameters that can be used to modify a Data Guard environment

### 4.2.6 Create a Windows Service

If the standby system is on a Windows system, then you must create a Windows Service. Run the ORADIM utility to create both the Windows Service and a password file. For example:

```
WINNT> oradim -NEW -SID payroll3 -STARTMODE auto
```

**See Also:** *Oracle9i Database Administrator's Guide for Windows* for more information about using the ORADIM utility

### 4.2.7 Configure the Listener for Both the Primary and Standby Databases

On both the primary and standby sites, use Oracle Net Manager to configure a listener for the respective databases. If you plan to manage the configuration using the Data Guard broker, you must configure the listener to use the TCP/IP protocol

and statically register service information for each database using the SID for the database instance.

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

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

**See Also:** *Oracle9i Net Services Administrator's Guide*

### 4.2.8 Enable Dead Connection Detection on the Standby System

Enable dead connection detection by setting the SQLNET.EXPIRE_TIME parameter to 2 in the SQLNET.ORA parameter file on the standby system. For example:

```
SQLNET.EXPIRE_TIME=2
```

### 4.2.9 Create Oracle Net Service Names

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

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

**See Also:** *Oracle9i Net Services Administrator's Guide*

### 4.2.10 Start and Mount the Logical Standby Database

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

```
SQL> STARTUP MOUNT PFILE=initpayroll3.ora;
```

### 4.2.11 Rename the Datafiles on the Logical Standby Database

On the logical standby database, rename all of the datafiles that were identified in Section 4.2.1 and copied from the primary database. For example:

```
SQL> ALTER DATABASE RENAME FILE '/disk1/oracle/oradata/payroll/system01.dbf'
  2> TO '/disk1/oracle/oradata/payroll/standby/system01.dbf';
```

```
SQL> ALTER DATABASE RENAME FILE '/disk1/oracle/oradata/payroll/undotbs01.dbf'
  2> TO '/disk1/oracle/oradata/payroll/standby/undotbs01.dbf'

SQL> ALTER DATABASE RENAME FILE '/disk1/oracle/oradata/payroll/cwmlite01.dbf'
  2> TO '/disk1/oracle/oradata/payroll/standby/cwmlite01.dbf'
.
.
.
```

These statements specify the datafile names that are in the control file, it does not rename the actual datafiles.

On the standby database, query the NAME column in the V$DATAFILE view to verify that all datafile locations are correct. (This is the same query shown in Section 4.2.1.)

### 4.2.12 Rename Online Redo Logs on the Logical Standby Database

Although the online redo logs were not copied from the primary database, they must all be renamed so that the pointer in the control file is updated to point to the correct location. The location and name of the online redo logs were identified in Section 4.2.1. For example:

```
SQL> ALTER DATABASE RENAME FILE '/disk1/oracle/oradata/payroll/redo01.log'
  2> TO '/disk1/oracle/oradata/payroll/standby/redo01.log';
```

On the standby database, query the NAME column in the V$DATAFILE view to verify that all log locations are correct. (This is the same query shown in Section 4.2.1.)

### 4.2.13 Turn On the Database Guard

To prevent users from updating objects in the logical standby database, turn on the database guard by issuing the following SQL statements on the standby database:

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

### 4.2.14 Reset the Database Name of the Logical Standby Database

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

Before you run the DBNEWID (nid) utility, you must shut down the database, and then start and mount it. For example:

```
SQL> SHUTDOWN IMMEDIATE;
SQL> STARTUP MOUNT PFILE=initpayroll3.ora;
```

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

```
nid TARGET=SYS/password@PAYROLL3 DBNAME=PAYROLL3
Connected to database PAYROLL (DBID=1456557175)

Control Files in database:
    /disk1/oracle/oradata/payroll/standby/stdby.ctl
Change database ID and database name PAYROLL to PAYROLL3? (Y/[N]) => y

Proceeding with operation
Changing database ID from 1456557175 to 416458362
Changing database name from PAYROLL to PAYROLL3
    Control File /disk1/oracle/oradata/payroll/standby/payroll3.ctl - modified
    Datafile /disk1/oracle/oradata/payroll/standby/system01.dbf - dbid changed,
wrote new name
    Datafile /disk1/oracle/oradata/payroll/standby/undotbs01.dbf -dbid changed,
wrote new name
    .
    .
    .
    Control File /disk1/oracle/oradata/payroll/standby/payroll3.ctl-dbid
changed, wrote new name

Database name changed to PAYROLL3.
Modify parameter file and generate a new password file before restarting.
Database ID for database PAYROLL3 change to 416458362.
All previous backups and archived redo logs for this database are unusable.
Shut down database and open with RESETLOGS option.
Successfully changed database name and ID.
DBNEWID - Completed successfully.
```

If you use a password file for authenticatison, you must re-create the password file after running the Oracle DBNEWID (nid) utility.

### 4.2.15 Change the Database Name in the Parameter File

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

**Step 1 Modify the DB_NAME parameter.**

Set the DB_NAME initialization parameter in the text initialization parameter file to match the new name:

```
.
.
.
db_name=PAYROLL3
.
.
.
```

**Step 2 Shut down the standby database.**

On the standby database, issue the following SQL statement:

```
SQL> SHUTDOWN IMMEDIATE;
```

**Step 3 Create a server parameter file for the standby database.**

Connect to an idle instance of the standby database, and create a server parameter file for the standby database from the text initialization parameter file that was edited in Section 4.2.5 and in step 1 of this section. For example:

```
SQL> CREATE SPFILE FROM PFILE=initpayroll3.ora;
```

**Step 4 Restart the logical standby database.**

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

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

## 4.2.16 Create a New Temporary File for the Logical Standby Database

The temporary files, which were included as a part of the closed backup operation on the primary database, are not viable on the logical standby database. (It is not necessary to copy temporary files from the primary database to the logical standby database.)

To identify and drop obsolete temporary files, perform the following steps on the logical standby database.

**Step 1 Identify the current temporary files.**

On the logical standby database, issue the following query to identify the current temporary files for the standby database:

```
SQL> SELECT * FROM V$TEMPFILE;
no rows selected
```

If this query returns "no rows selected" as shown in the example, skip step 2 and go to step 3.

**Step 2 Drop each current temporary file from the standby database.**

Drop each current temporary file from the standby database, as follows:

```
SQL> ALTER DATABASE TEMPFILE 'tempfilename' DROP;
```

**Step 3 Add a new temporary file.**

On the logical standby database, perform the following tasks to add a new temporary file to the tablespace:

1. Identify the tablespace that should contain the temporary file, for example:

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

TABLESPACE_NAME
--------------------------------
TEMP
```

2. Add a new temporary file, for example:

```
SQL> ALTER TABLESPACE TEMP ADD TEMPFILE
  2> '/disk1/oracle/oradata/payroll/standby/temp01.dbf'
  3> SIZE 40M REUSE;
```

## 4.2.17 Register the Archived Redo Log and Start SQL Apply Operations

To register the most recently archived redo log and begin applying data from the redo logs to the standby database, perform the following steps.

**Step 1 Register the most recently archived redo log with log apply services.**

Register the archived redo log that was identified in step 8 of Section 4.2.2. The following example specifies the filename and location of the most recently archived redo log that you copied to the logical standby site:

```
SQL> ALTER DATABASE REGISTER LOGICAL LOGFILE
  2> '/disk1/oracle/oradata/payroll/standby/arc0004.001';
```

**Step 2 Start applying redo logs to the logical standby database.**

Specify the following SQL statement to begin applying redo logs to the logical standby database. For example:

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

---

**Note:** Only include the INITIAL keyword the first time you start applying data from redo logs to the standby database. For example, the following statements show how to subsequently stop and start SQL apply operations:

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

---

## 4.2.18 Enable Archiving to the Logical Standby Database

This section describes the minimum amount of work you must do on the primary database to set up and enable archiving on the logical standby database.

> **See Also:** Chapter 5 for information about log transport services and Chapter 12 for reference information about additional attributes you can set on the LOG_ARCHIVE_DEST_*n* initialization parameter

**Step 1 Set initialization parameters to define archiving.**

To configure archive logging from the primary database to the standby site the LOG_ARCHIVE_DEST_*n* and LOG_ARCHIVE_DEST_STATE_*n* parameters must be defined.

The following example sets the initialization parameters needed to enable archive logging on the standby site:

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_3='SERVICE=payroll3' SCOPE=BOTH;
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_3=ENABLE SCOPE=BOTH;
```

**Step 2 Start remote archiving.**

Archiving of redo logs to the remote standby location does not occur until after a log switch. A log switch occurs, by default, when an online redo log becomes full. To force the current redo logs to be archived immediately, use the SQL ALTER SYSTEM statement on the primary database. For example:

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

**See Also:** Chapter 7 if you intend for the logical standby database to be the target of a switchover operation in the future. This chapter describes how to define a database link to the primary database that will be used during switchover operations.

## 4.3 Verify the Logical Standby Database

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

### Step 1 Verify that the redo logs have been registered.

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

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

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

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

8 rows selected.
```

### Step 2 Archive some redo logs.

Connect to the primary database and archive some redo logs. For example:

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

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

### Step 3 Query the DBA_LOGSTDBY_LOG view again.

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

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

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

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

10 rows selected.
```

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

### Step 4 Verify that data from the redo logs is being applied correctly.

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

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

NAME                           VALUE
```

```
----------------------------- -----------------------------
coordinator state             INITIALIZING
```

In the example, the output from the DBA_LOGSTDBY_STATS view shows the coordinator process is in the initialization state. When the coordinator process is initializing, log apply services are preparing to begin SQL apply operations, but data from the redo logs is not being applied to the logical standby database.

> **Note:** The first time log apply services start, it can take a considerable amount of time for log apply services to initialize and prepare the database. If the logical standby database has many tables, the initialization and preparation can take hours. However, after the initial preparation activity, subsequent restarts go much more quickly.

Knowing the state of the coordinator process is of particular importance because it is the LSP background process that instructs all of the other logical standby processes.

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

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

Example 4–2 shows typical output during the initialization phase.

***Example 4–2 V$LOGSTDBY Output During the Initialization Phase***

```
SQL> COLUMN STATUS FORMAT A50

SQL> COLUMN TYPE FORMAT A12

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

SQL> SELECT TYPE, HIGH_SCN, STATUS FROM V$LOGSTDBY;
```

```
TYPE           HIGH_SCN STATUS
------------ ---------- --------------------------------------------------
COORDINATOR             ORA-16126: loading table or sequence object number
READER                  ORA-16116: no work available
BUILDER                 ORA-16116: no work available
PREPARER                ORA-16116: no work available
```

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

Example 4–3 shows typical output during the applying phase. Notice that the values in the HIGH_SCN column continue to increment. The numbers in this column will continue to increase as long as changes are being applied. The HIGH_SCN column serves only as an indicator of progress.

***Example 4–3 V$LOGSTDBY Output During the Applying Phase***

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

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

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

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

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

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

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

11 rows selected.
```

### Step 6 Check the overall progress of log apply services.

To check the overall progress of log apply services, query the DBA_LOGSTDBY_PROGRESS view on the standby database. For example:

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

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

When the numbers in the APPLIED_SCN and NEWEST_SCN columns are equal (as shown in the query example), it means that all of the available data in the redo log was applied. These values can be compared to the values in the FIRST_CHANGE# column in the DBA_LOGSTDBY_LOG view to see how much log information has to be applied and how much remains.

> **See Also:** Section 5.9, "Monitoring Redo Log Archival Information" and Section 6.5, "Monitoring Log Apply Services for Physical Standby Databases" for information about how to verify that both log transport and log apply services are working correctly

# 5

# Log Transport Services

This chapter describes log transport services and how they control the transmission of redo data to standby databases. It includes the following topics:



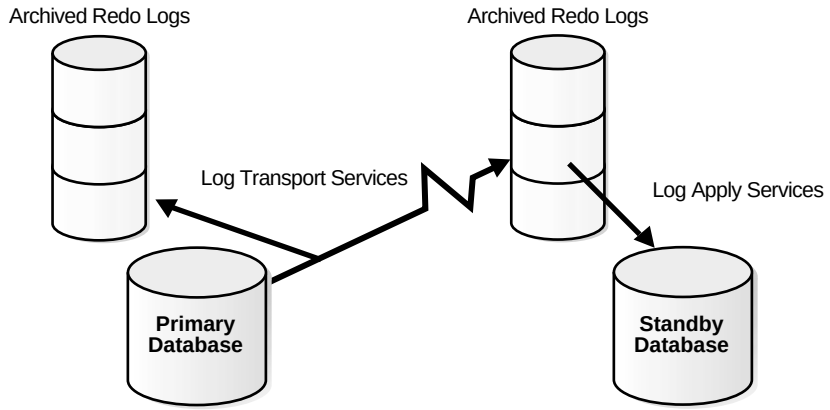
## 5.1 Introduction to Log Transport Services

**Log transport services** control the automated transfer of redo data within a Data Guard configuration.

Log transport services also control the level of data protection for your database. You can configure log transport services to balance data protection and availability against performance. In a Data Guard environment, log transport services coordinate with log apply services and role management services for switchover and failover operations.

Figure 5–1 shows a simple Data Guard configuration with redo logs being archived from a primary database to a local destination and to a remote standby database destination using log transport services.

***Figure 5–1 Archiving Redo Logs***




The following concepts are important in understanding log transport services:

- Redo logs

  Redo logs contain the data needed to recover a database. They are also used on a standby system to apply updates to the standby database.

- Redo log destinations

  Redo log destinations specify the location and types of redo logs along with the policies used to manage them.

- Transmission and reception of redo logs

  Log transport services are responsible for the transmission and reception of redo data. This involves transmitting redo data throughout a Data Guard configuration and ensuring that data from the redo logs is committed to disk.

- Data protection

  You can set archive destination attributes and log transport services options to enforce any of the three distinct modes of data protection.

In summary, log transport services transmit redo logs to various destinations where redo data is written to archived redo logs.

## 5.2 Data Protection Modes

A Data Guard configuration always runs in one of three data protection modes: maximum protection, maximum availability, or maximum performance. Each of these protection modes provides a different balance of data protection, data availability, and primary database performance. To select the protection mode that best meets your business needs, you should carefully consider your data protection requirements and the performance expectations of your users.

**Maximum protection** mode offers the highest level of data protection. A primary database transaction will not commit until the redo data needed to recover that transaction is written to at least one physical standby database that meets the minimum requirements for this mode. If the primary database is unable to write the redo data to at least one such standby database, the primary database will shut down to prevent the generation of unprotected data. This protection mode guarantees no data loss, but it has the highest potential impact on the performance and availability of the primary database.

**Maximum availability** mode offers the next highest level of data protection. A primary database transaction will not commit until the redo data needed to recover that transaction is written to at least one standby database that meets the minimum requirements for this mode. Unlike maximum protection mode, the primary database will not shut down if it is unable to write the redo data to at least one such standby database. Instead, the protection mode will be temporarily lowered to maximum performance mode until the fault is corrected and the standby database catches up with the primary database. This mode guarantees no data loss unless the primary database fails while it is in maximum performance mode. This protection mode provides the highest level of data protection that is possible without affecting the availability of the primary database.

**Maximum performance** mode is the default protection mode. A primary database transaction will not wait to commit until the redo data needed to recover that transaction is written to a standby database. Therefore, some data might be lost if the primary database fails and the redo data needed to recover committed transactions is not available at any standby database. This mode provides the highest level of data protection that is possible without affecting the performance or availability of the primary database.

Each of these data protection modes requires that at least one standby database in the configuration use a specific set of log transport services attributes. The remainder of this chapter describes those attributes in detail. Once you understand these attributes, you can make the appropriate configuration changes needed to support the Data Guard protection mode that is right for your business.

See Section 5.7 for a description of the SQL statement that is used to set the data protection mode for a Data Guard configuration.

## 5.3 Transporting Redo Data

Data Guard automatically maintains the standby database by transmitting primary database redo data to the standby system and then applying the redo logs to the standby database. This section describes using the following types of redo logs in a Data Guard configuration:

- Online Redo Logs
- Archived Redo Logs
- Standby Redo Logs

### 5.3.1 Online Redo Logs

The **online redo logs** are a set of two or more files that record all changes made to Oracle datafiles and control files. Whenever a change is made to the database, the Oracle database server writes the data and generates a redo record in the redo buffer. The logwriter process flushes the contents of the redo buffer into the online redo log.

The current online redo log is the one being written to by the logwriter process. When the logwriter process gets to the end of the file, it performs a log switch and begins writing to a new log file. If you run the database in ARCHIVELOG mode, then an archiver process copies the online redo log into an archived redo log.

A **redo log group** is a set of two or more redo logs that are multiplexed for redundancy. The logwriter process will write the same redo data to all redo logs in a group. If a write error occurs on one of the logs, then the redo data will still be available in the other redo logs in the group.

Both the size of the online redo logs and the frequency with which they switch affect the generation of archived redo logs at the primary site. In general, the most important factor in deciding what size to make an online redo log is the amount of application data that needs to be applied to a standby database during a database failover operation. The larger the online redo log, the more data needs to be applied to a standby database to make it consistent with the primary database.

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

**See Also:** *Oracle9i Database Administrator's Guide* for more details about configuring online redo logs and online redo log groups

## 5.3.2 Archived Redo Logs

An **archived redo log** is a copy of one of the filled members of an online redo log group made when the database is in ARCHIVELOG mode. After the LGWR process fills each online redo log with redo records, the archiver process copies the log to one or more archive log destinations.

By archiving filled online redo logs, older redo log data is preserved for operations such as media recovery, while the preallocated online redo logs continue to be reused to store the most current database changes. On a standby system, the archived redo logs are used to apply primary database changes to the standby database.

### 5.3.2.1 Setting Permission to Archive Redo Logs

Permission for the archiving of online redo logs to remote destinations is specified using the REMOTE_ARCHIVE_ENABLE initialization parameter. This parameter provides TRUE, FALSE, SEND, and RECEIVE options. In most cases, you should set this parameter to TRUE on both the primary and standby databases in a Data Guard environment. To independently enable and disable the sending and receiving of remote archived redo logs, use the SEND and RECEIVE values.

For example, to ensure that the primary database never accidentally receives any archived redo logs, you can set the REMOTE_ARCHIVE_ENABLE initialization parameter to SEND on the primary database. Conversely, to ensure that the standby database never remotely archives the standby redo logs, you can set the REMOTE_ARCHIVE_ENABLE initialization parameter to RECEIVE on the standby database.

**See Also:** Chapter 7 for information about setting initialization parameters for role transition operations

### 5.3.2.2 Controlling the Reuse of Archived Redo Logs

The CONTROL_FILE_RECORD_KEEP_TIME initialization parameter specifies the minimum number of days that must pass before a reusable record in the control file can be reused. Setting this parameter prevents log transport services from overwriting a reusable record in the control file. (It applies only to records in the control file that are serially reusable.) This parameter helps to ensure that the archived redo log information remains available on the standby database. This is especially important when you have specified an apply delay for the standby

database. The range of values for this parameter is 0 to 365 days. The default value is 7 days.

> **See Also:** *Oracle9i Database Reference* for more details about the CONTROL_FILE_RECORD_KEEP_TIME initialization parameter

#### 5.3.2.3 Specifying a Time Lag for the Application of Redo Logs

In some cases, you may want to create a time lag between the archiving of a redo log at the primary site and the applying of the redo log at the standby site. A time lag can protect against the application of corrupted or erroneous data from the primary site to the standby site.

Use the DELAY=*minutes* attribute of the LOG_ARCHIVE_DEST_*n* initialization parameter to specify a time lag for applying redo logs at the standby site. The DELAY interval is relative to when the archived redo log is complete at the destination. It does not delay the transport of the redo log to the standby database. The default setting for this attribute is NODELAY. If the DELAY attribute is set with no value specified, then the value for this attribute is 30 minutes.

> **See Also:**
>
> - Section 10.2, "Using a Physical Standby Database with a Time Lag"
> - Section 13.12 for physical standby databases using the DELAY control option on the ALTER DATABASE RECOVER MANAGED STANDBY DATABASE DELAY statement to supersede any apply delay interval specified on the primary database
> - *Oracle9i Supplied PL/SQL Packages and Types Reference* for logical standby databases using the DBMS_LOGSTDBY.APPLY_SET procedure to supersede any apply delay interval specified on the primary database

### 5.3.3 Standby Redo Logs

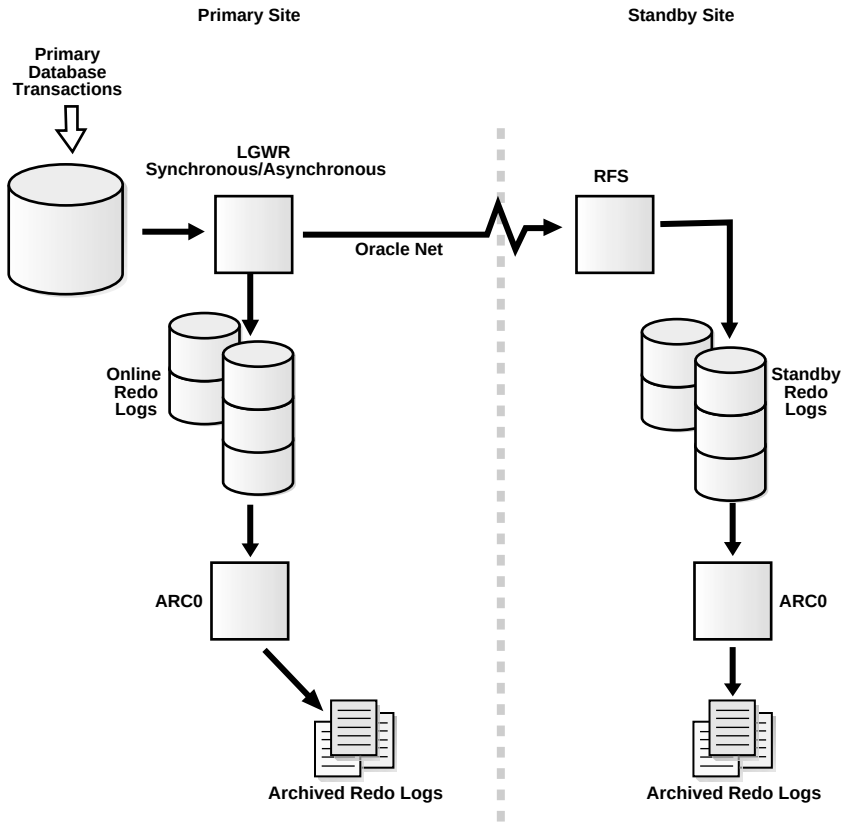
> **Note:** This section applies to physical standby databases only.

Standby redo logs are similar to online redo logs and are required for physical standby databases running in maximum protection mode and maximum availability mode. Redo data transmitted from the primary database is received by

the remote file server process (RFS) on the standby system where the RFS process will write the redo data to either standby redo logs or to archived redo logs.

Standby redo logs form a separate pool of log file groups. During a failover operation, they enable Data Guard to apply more redo data than what is available in the archived redo logs alone. Because standby redo logs must be archived before the data can be applied to the standby database, the archiver process must be started on the standby database. Figure 5–2 shows a Data Guard configuration in which the RFS process receives redo data from the log writer process and writes it to standby redo logs. A log switch on the primary database triggers a log switch on the standby database that results in the archiver process archiving the standby redo logs to archived redo logs on the standby database.

***Figure 5–2 Redo Log Reception Options***



#### 5.3.3.1 Size and Number of Standby Redo Logs

The size of a standby redo log must exactly match the primary database online redo logs. For example, if the primary database uses two online redo log groups whose log size is 100K and 200K, respectively, then the standby database should have standby redo log groups with those same sizes.

**5.3.3.1.1 Number of Standby Redo Log Groups** The minimum configuration should have one more standby redo log group than the primary database.

It might be necessary to create additional standby log groups on the physical standby database, so that the archival operation has time to complete before the standby redo log is reused by the RFS process. If the primary database is operating in maximum protection mode and a standby redo log cannot be allocated, the primary database instance might shut down immediately. If the primary database is operating in maximum protection mode or maximum availability mode, then the primary database might wait for the standby redo log to become available. Therefore, be sure to allocate an adequate number of standby redo logs.

> **Caution:** Whenever you add an online redo log to the primary database, you must add a corresponding standby redo log to the standby database. If you do not add a standby redo log to the standby database, the primary database might shut down.

During testing, the easiest way to determine if the current standby log configuration is satisfactory is to examine the contents of the RFS process trace file and the database alert log. If messages indicate that the RFS process frequently has to wait for a group because archiving did not complete, add more standby log groups.

When you use Real Application Clusters, the various standby redo logs are shared among the various primary database instances. Standby redo log groups are not dedicated to a particular primary database thread.

**5.3.3.1.2 Guidelines for Standby Redo Log Groups** Consider the database parameters that can limit the number of standby redo log groups before setting up or altering the configuration of the standby redo log groups. The following parameters limit the number of standby redo log groups that you can add to a database:

- The `MAXLOGFILES` clause of the `CREATE DATABASE` statement for the primary database determines the maximum number of groups of standby redo logs per physical standby database. The only way to override this limit is to re-create the primary database or control file.

- The LOG_FILES parameter can temporarily decrease the maximum number of groups of standby redo logs for the duration of the current instance.
- The MAXLOGMEMBERS clause of the CREATE DATABASE statement used for the primary database determines the maximum number of members per group. The only way to override this limit is to re-create the primary database or control file.

**See Also:** *Oracle9i SQL Reference*

#### 5.3.3.2 Creating Standby Redo Logs

Standby redo logs are created using the ADD STANDBY LOGFILE clause of the ALTER DATABASE statement.

To verify that standby redo logs were created, query the V$STANDBY_LOG view (displays standby redo log status as ACTIVE or INACTIVE) or the V$LOGFILE view. The following example queries the V$LOGFILE view:

```
SQL> SELECT * FROM V$LOGFILE WHERE TYPE = 'STANDBY';
```

#### 5.3.3.3 Creating Standby Redo Log Groups

---

**Note:** Although standby redo logs are only used when the database is running in the physical standby role, Oracle Corporation recommends that you create standby redo logs on the primary database so that the primary database can switch over quickly to a standby role without the need for DBA intervention.

---

Standby redo logs can be multiplexed to increase the availability of redo logs, similar to the way that online redo logs are multiplexed. Plan the standby redo log configuration of a database and create all required groups and members of groups after you instantiate the standby database. To create new standby redo log groups and members, you must have the ALTER DATABASE system privilege. A database can have as many groups as the value of the MAXLOGFILES clause that was specified on the SQL CREATE DATABASE statement.

To create a new group of standby redo logs, use the ALTER DATABASE statement with the ADD STANDBY LOGFILE clause.

The following statement adds a new group of standby redo logs to a physical standby database:

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

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

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

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

The physical standby database begins using the newly created standby redo logs the next time there is a log switch on the primary database. To verify that the standby redo log groups are created and running correctly, invoke a log switch on the primary database, and then query the V$STANDBY_LOG view on the physical standby database.

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

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

#### 5.3.3.4 Adding Standby Redo Log Members to an Existing Group

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

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

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

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

## 5.4 Destination Parameters and Attributes

Log transport services transmit redo data to up to 10 redo log destinations. You configure the primary database to perform archiving using the LOG_ARCHIVE_DEST_*n* (where *n* is an integer from 1 to 10) initialization parameter and corresponding LOG_ARCHIVE_DEST_STATE_*n* (where *n* is an integer from 1 to 10) initialization parameter. You can also use these initialization parameters to set up cascading standby databases, as described in Appendix D.

There are a several initialization parameters that are used to configure destinations. Some parameters, such as LOG_ARCHIVE_DEST_*n* might have several attributes that further refine the meaning of the parameter.

Archive destination attributes specify all aspects of destinations, not just the location. Particularly, they specify the following properties:

- Location of the destination.
- Redo log transmission and reception characteristics of the destination.
- Relationships between destinations such as dependencies.
- Importance of a destination. For example, the destination might be optional.
- Time delay for applying redo logs to the standby database.
- Error handling and retransmission of redo logs.

The parameters related to archive destinations follow:

- LOG_ARCHIVE_DEST_*n*

  Controls most of the behavior and properties of the destination; this parameter has many attributes. Refer to Chapter 12 for a full description of all of the LOG_ARCHIVE_DEST_*n* attributes.

- LOG_ARCHIVE_DEST_STATE_*n*

  Controls state of the destination. For each LOG_ARCHIVE_DEST_*n* parameter, there is a corresponding LOG_ARCHIVE_DEST_STATE_*n* parameter.

- STANDBY_ARCHIVE_DEST

  Determines the location of archived redo logs on the standby database.

- LOG_ARCHIVE_FORMAT

  Specifies the format for archived redo log filenames. STANDBY_ARCHIVE_DEST and LOG_ARCHIVE_FORMAT are concatenated to generate fully-qualified standby database archived redo log filenames.

- LOG_ARCHIVE_MIN_SUCCEED_DEST

  Defines the minimum number of local destinations that must receive redo logs successfully before the log writer process on the primary database can reuse the online redo logs.

- REMOTE_ARCHIVE_ENABLE

  Enables or disables the sending of redo logs to remote destinations and the receipt of remote redo logs

## 5.4.1 Specifying Archive Destinations for Redo Logs

In addition to setting up the primary database to run in ARCHIVELOG mode, you must configure the primary database to archive redo logs by setting destinations and associated states. You do this using the LOG_ARCHIVE_DEST_*n* initialization parameter and corresponding LOG_ARCHIVE_DEST_STATE_*n* parameter.

The LOG_ARCHIVE_DEST_STATE_*n* (where *n* is an integer from 1 to 10) initialization parameter specifies the *state* of the corresponding destination indicated by the LOG_ARCHIVE_DEST_*n* initialization parameter (where *n* is the same integer). For example, the LOG_ARCHIVE_DEST_STATE_3 parameter specifies the state of the LOG_ARCHIVE_DEST_3 destination.

Table 5–1 describes the LOG_ARCHIVE_DEST_STATE_*n* parameter attributes.

***Table 5–1 LOG_ARCHIVE_DEST_STATE_n Initialization Parameter Attributes***

| Attribute | Description |
|---|---|
| ENABLE | Log transport services can archive redo logs at this destination. |
| DEFER | Log transport services will not archive redo logs to this destination. This is an unused destination. |
| ALTERNATE | This destination is not enabled, but it will become enabled if communication to another destination fails. |

To set up log transport services to archive redo logs to the standby database named payroll2 on a remote node, make the following modifications to the primary database initialization parameter file. These modifications will take effect after the next log switch. For example:

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_2='SERVICE=payroll2';
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2=ENABLE;
```

## 5.4.2 Specifying Storage Locations for Archived Redo Logs and Standby Redo Logs

Use the STANDBY_ARCHIVE_DEST initialization parameter on the standby database to specify the directory in which to store the archived redo logs. Log transport services use this value in conjunction with the LOG_ARCHIVE_FORMAT parameter to generate the archived redo log filenames on the standby site.

| Parameter | Indicates | Example |
|---|---|---|
| STANDBY_ARCHIVE_DEST | Directory in which to place archived online redo logs | STANDBY_ARCHIVE_DEST= /arc_dest/ |
| LOG_ARCHIVE_FORMAT | Format for filenames of archived online redo logs | LOG_ARCHIVE_FORMAT = "log%d_%t_%s.arc"<br>**Note:** The %d corresponds to the database ID, and %s corresponds to the sequence number. The %t, which is required for Real Application Clusters configurations, corresponds to the thread. |

Log transport services store the fully-qualified filenames in the standby control file. Log apply services use this information to perform recovery operations on the standby database. The following example shows how to set the LOG_ARCHIVE_FORMAT initialization parameter:

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_FORMAT='log%d_%t_%s.arc';
```

Issue the following query on the primary database to display the list of archived redo logs that are on the standby system:

```
SQL> SELECT NAME FROM V$ARCHIVED_LOG;
NAME
--------------------------------------------------------------------------------
/arc_dest/log_1_771.arc
/arc_dest/log_1_772.arc
/arc_dest/log_1_773.arc
/arc_dest/log_1_774.arc
/arc_dest/log_1_775.arc
```

When standby redo logs are used, the LOG_ARCHIVE_DEST_*n* initialization parameter (where *n* is a value from 1 to 10) on the standby database specifies the directory in which to archive standby redo logs.

| Parameter | Indicates | Example |
|---|---|---|
| LOG_ARCHIVE_DEST_*n* | The directory for storage of archived redo logs on the standby site | LOG_ARCHIVE_DEST_1= 'LOCATION=/oracle/stby/arc/'<br>**Note:** If you do not define this parameter, the value of the STANDBY_ ARCHIVE_DEST parameter is used. |
| LOG_ARCHIVE_FORMAT | Format for filenames of archived online redo logs | LOG_ARCHIVE_FORMAT = "log%d_%t_%s.arc"<br>**Note:** The %d corresponds to the database ID, and %s corresponds to the sequence number. The %t, which is required for Real Application Clusters configurations, corresponds to the thread. |

**Note:** When using standby redo logs, you must enable the archiver process (ARC*n*) on the standby database. Oracle Corporation recommends that you always set the LOG_ARCHIVE_ START initialization parameter to TRUE on the standby database.

## 5.4.3 Specifying Mandatory and Optional Destinations

You can specify a policy for reuse of online redo logs using the attributes OPTIONAL or MANDATORY with the LOG_ARCHIVE_DEST_*n* parameter. Oracle Corporation recommends that you set remote destinations to OPTIONAL. (This is the default.) The archival operation of an optional destination can fail, and the online redo logs are overwritten. If the archival operation of a mandatory destination fails, online redo logs cannot be overwritten.

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

Example 5–1 shows how to set a mandatory local archiving destination and enable that destination.

***Example 5–1 Setting a Mandatory Archiving Destination***

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

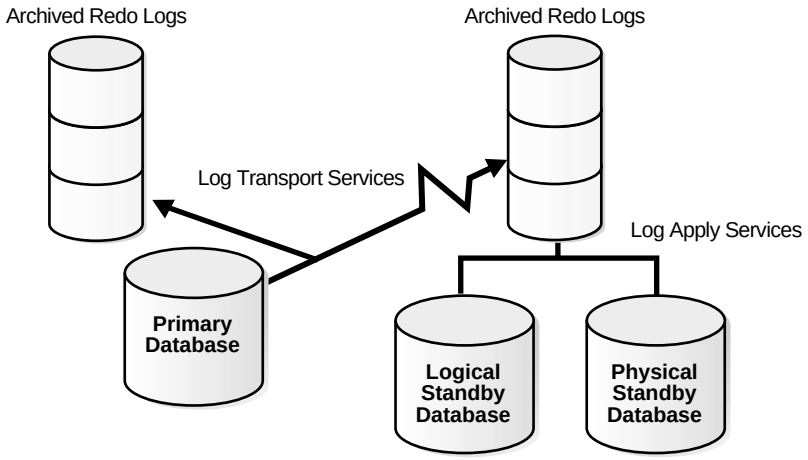
### 5.4.4 Sharing a Log File Destination Among Multiple Standby Databases

Archiving redo logs to a remote database can be defined as being dependent upon the success or failure of an archival operation for another destination. This is known as a dependent destination.

Use the DEPENDENCY attribute of the LOG_ARCHIVE_DEST_*n* initialization parameter to define a dependent destination. This attribute indicates that this destination depends on the successful completion of archival operations for the parent destination.

Figure 5–3 shows a Data Guard configuration in which the primary database transports redo data to one archiving destination that acts as a shared destination for both a logical standby database and a physical standby database.

***Figure 5–3 Data Guard Configuration with Dependent Destinations***




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

- When you configure a physical standby database and a logical standby database on the same node.
- The standby database and the primary database are on the same node. Therefore, the archived redo logs are implicitly accessible to the standby database.
- Operating system-specific network file systems are used, providing remote standby databases with access to the primary database archived redo logs.

- Mirrored disk technology is used to provide transparent networking support across geographically remote distances.
- There are multiple standby databases on the same remote node, sharing access to common archived redo logs for staggered managed recovery operations.

In these situations, although a physical archival operation is not required, the standby database needs to know the location of the archived redo logs. This allows the standby database to access the archived redo logs when they become available for application by log apply services. You must specify an archiving destination as being dependent on the success or failure of another (parent) destination.

## 5.4.5 Specifying Archive Failure Policies

Use the REOPEN and MAX_FAILURES attributes of the LOG_ARCHIVE_DEST_*n* initialization parameter to specify what actions are to be taken when archiving to a destination fails. These actions include:

- Retrying the archiving operation to a failed destination after a specified period of time, up to a limited number of times
- Using an alternate or substitute destination

Use the REOPEN attribute of the LOG_ARCHIVE_DEST_*n* parameter to determine if and when the archiver process or the log writer process attempts to archive redo logs again to a failed destination following an error.

Use the REOPEN=*seconds* attribute to specify the minimum number of seconds that must elapse following an error before the archiving process will try again to access a failed destination. The default value is 300 seconds. The value set for the REOPEN attribute applies to all errors, not just connection failures. You can turn off the option by specifying NOREOPEN, which will prevent the destination from being retried after a failure occurs.

You can use the REOPEN attribute, in conjunction with the MAX_FAILURE attribute, to limit the number of consecutive attempts that will be made to reestablish communication with a failed destination. Once the specified number of consecutive attempts is exceeded, the destination is treated as if the NOREOPEN attribute was specified.

The REOPEN attribute is required when you use the MAX_FAILURE attribute. Example 5–2 shows how to set a retry time of 5 seconds and limit retries to 3 times.

***Example 5–2 Setting a Retry Time and Limit***

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

### 5.4.6 Other Destination Types

There are four types of remote destinations: physical standby databases, logical standby databases, archive log repositories, and cross-instance archival database environments. The more common destinations, physical and logical standby databases, are described in Chapter 1. The following list describes some additional destinations:

- Archive log repository

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

- Cross-instance archival database environment

  A **cross-instance archival** database environment is possible on both the primary and standby databases. Within a Real Application Clusters environment, each instance directs its archived redo logs to a single instance of the cluster. This instance, known as the **recovery instance,** is typically the instance where managed recovery is performed. The recovery instance typically has a tape drive available for RMAN backup and restoration support.

## 5.5 Transmission and Reception of Redo Data

Log transport services automatically transmit and receive all redo logs in a Data Guard configuration. You can tailor the characteristics of transmission and reception to balance data protection levels against performance.

Archiving redo logs to a remote destination requires uninterrupted connectivity through Oracle Net. If the destination is a remote physical standby database, a physical standby database must be mounted or open in read-only mode to receive the archived redo logs. A logical standby database must be open.

You can specify the following with respect to log transport services:

- Process that transmits redo logs
- Network transmission mode of redo logs
- How data from the redo logs is written to disk

### 5.5.1 Specifying the Process that Transmits Redo Data

To minimize data loss in the event of a primary database failure, you want to copy data from the primary database to the standby database as it is being generated. You can choose to have either the log writer process or the archiver process transmit redo logs to a destination.

To specify which process transmits redo data, use either the ARCH or LGWR attribute of the LOG_ARCHIVE_DEST_*n* initialization parameter.

| Attribute | Example | Default |
|---|---|---|
| {ARCH \| LGWR} | LOG_ARCHIVE_DEST_3='SERVICE=stby1 LGWR' | ARCH |

The LGWR and ARCH attributes are mutually exclusive. Therefore, you cannot specify both attributes for the same destination. However, you can specify either the LGWR or the ARCH attribute for individual destinations. This allows you to choose the log writer process to transmit redo data for some destinations, while the archiver process transmits redo data to other destinations.

Choosing the ARCH attribute indicates that an archiver process (ARC*n*) will archive the current redo logs to the associated destination when a redo log switch occurs on the primary database. This is the default setting.

Choosing the LGWR attribute indicates that the log writer process (LGWR) will transmit redo data to the associated destination as it is generated. As redo is generated for the primary database, it is also propagated to the standby system where the RFS process writes the redo to either a standby redo log or to a standby archived redo log.

### 5.5.2 Specifying Network Transmission Mode

The only way to ensure you do not have any data loss is to write redo data to the standby database before it is committed on the primary database. If you specify the SYNC attribute, all network I/O operations are performed synchronously, in conjunction with each write operation to the online redo log. The transaction is not committed on the primary database until the redo data necessary to recover that transaction is received by the destination.

When you use the log writer process to archive redo logs, you can specify synchronous (SYNC) or asynchronous (ASYNC) network transmission of redo logs to archiving destinations using the SYNC or ASYNC attributes. If you do not specify

either the SYNC or ASYNC attribute, the default is the SYNC network transmission mode. Each of these transmission methods is described in the following list:

- SYNC network transmission method

  The SYNC attribute has the potential to affect primary database performance adversely, but provides the highest degree of data protection at the destination site. Synchronous transmission is required for no data loss environments.

- ASYNC network transmission method

  If you specify the ASYNC attribute, all network I/O operations are performed asynchronously, and control is returned to the executing application or user immediately. You can specify a block count to determine the size of the SGA network buffer to be used. Block counts from 0 to 20,480 are allowed. The attribute allows the optional suffix value K to represent 1,000 (the value 1K indicates 1,000 512-byte blocks). In general, for slower network connections, use larger block counts.

  **See Also:** Chapter 12, "LOG_ARCHIVE_DEST_n Parameter Attributes"

### 5.5.3 Writing Redo Data to Disk

Use the [NO]AFFIRM attribute of the LOG_ARCHIVE_DEST_*n* initialization parameter to specify if log archiving disk write I/O operations are to be performed synchronously or asynchronously.

---

**Note:** The AFFIRM and NOAFFIRM attributes apply only to online archived log destinations and has no effect on online redo log disk I/O operations.

---

## 5.6 Log Transport Services in Sample Configurations

Log transport services transport redo data to the systems in a Data Guard configuration. The processes used to transport redo data include the following:

- Log writer (LGWR)

  The log writer process collects transaction redo data on the primary database and updates the online redo logs. Furthermore, LGWR can transmit online redo data directly to standby systems.

- Archiver (ARCn)

The archiver process copies both online redo logs and standby redo logs to archive destinations. Archive destinations can be either local or remote. The archiver process runs on both primary and standby systems.

- Remote File Server (RFS)

  The remote file server runs on the standby system and receives redo data over the network from both LGWR and ARC*n*. The RFS process will write the redo data to either a standby redo log or to a standby archived redo log.

- Fetch archive log (FAL)

  The fetch archive log process helps to resolve gaps in archived redo logs. If a physical standby database detects that it is missing a redo log, the local FAL client will fetch the log.

The following figures show how log transport services work in various configurations.

Figure 5–4 shows the simplest configuration with a single local destination. The log writer process writes redo data to online redo logs. When each online redo log is filled, a log switch occurs and the archiver process archives the filled online redo log to an archived redo log. The filled online redo log is now available for reuse.

***Figure 5–4 Primary Database Archiving When There Is No Standby Database***

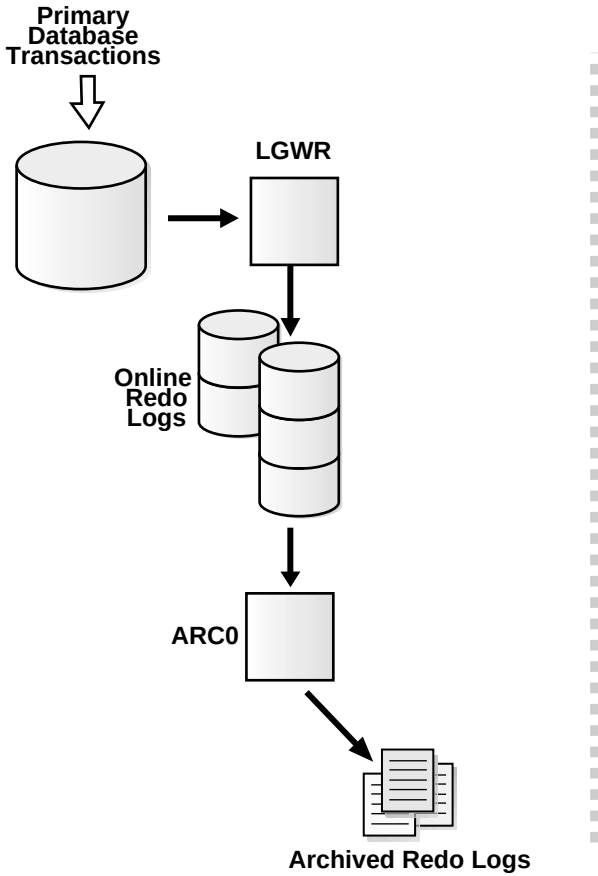


Figure 5–5 shows a Data Guard configuration with a local destination and a standby destination. At log switch time, the archiver process archives to both the local destination and the standby destination. Note that the archiver process uses Oracle Net to send redo data over the network to the RFS process. The RFS process writes the redo data to archived redo logs on the standby database. This figure also shows that themanaged recovery process (MRP) or logical standby process (LSP) is used to apply the redo logs to the standby database.

**See Also:** Chapter 6 for MRP and LSP process information

***Figure 5–5 Basic Data Guard Configuration***

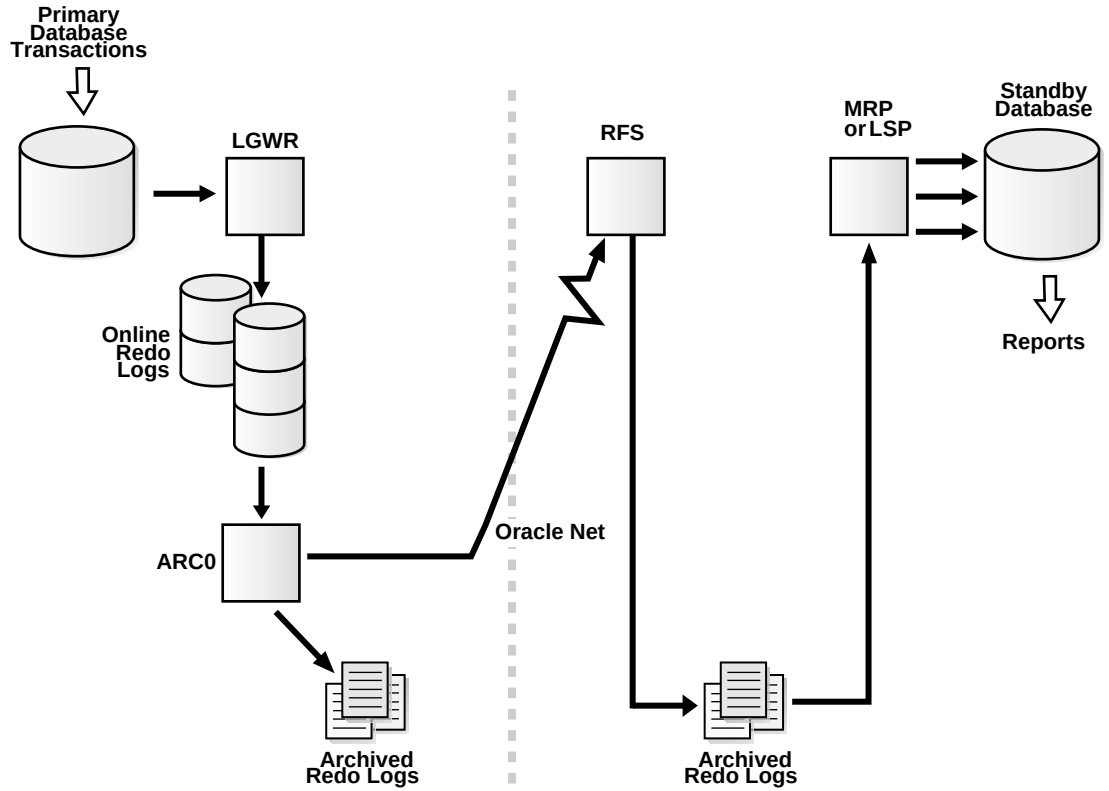


Figure 5–6 shows a Data Guard configuration with a local destination and a standby destination. In this configuration, the archiver on the primary system is archiving only to the local destination. Notice that the logwriter process is sending redo data to the standby system at the same time it is writing the data to the online redo log. The RFS process writes the redo data to an online redo log on the standby database. A log switch on the primary database triggers a log switch on the standby database, which causes the archiver process on the standby database to archive the redo logs to archived redo logs on the standby database. This configuration is a prerequisite for the highest level of data protection. For physical standby databases, Oracle Corporation recommends that you use standby redo logs. Standby redo logs are not supported for logical standby databases.

***Figure 5–6 Archiving to a Physical Standby Destination Using the Logwriter Process***

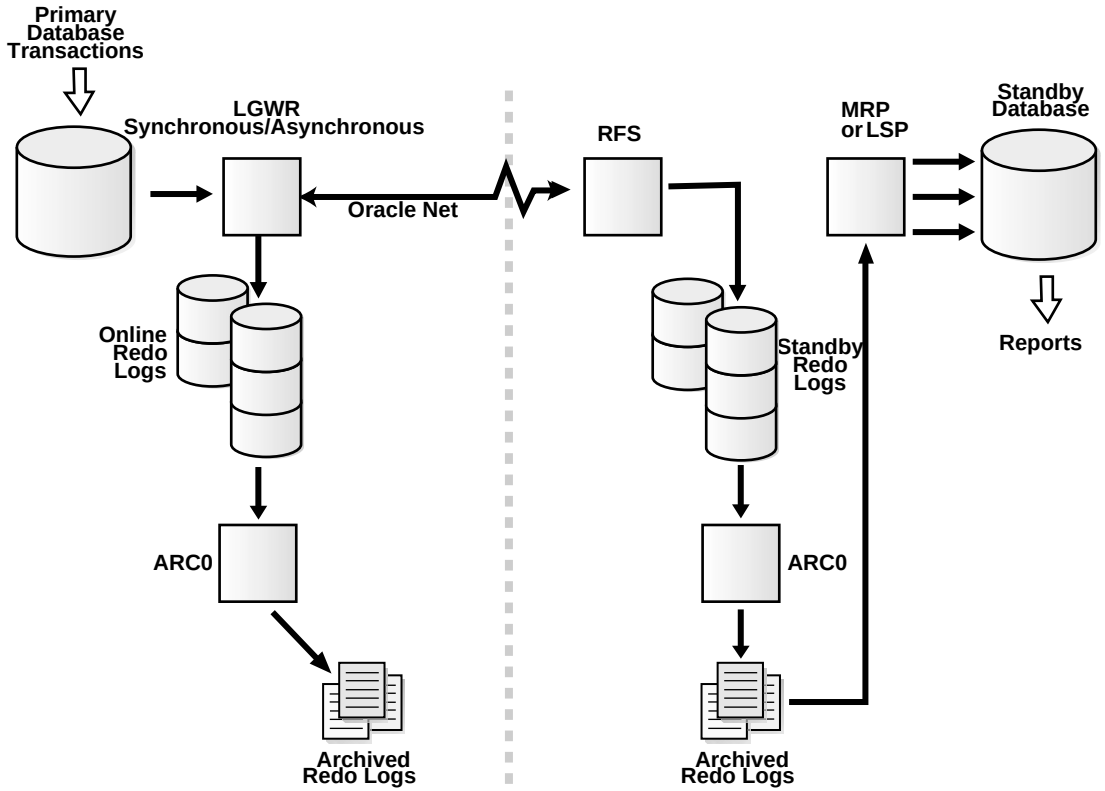
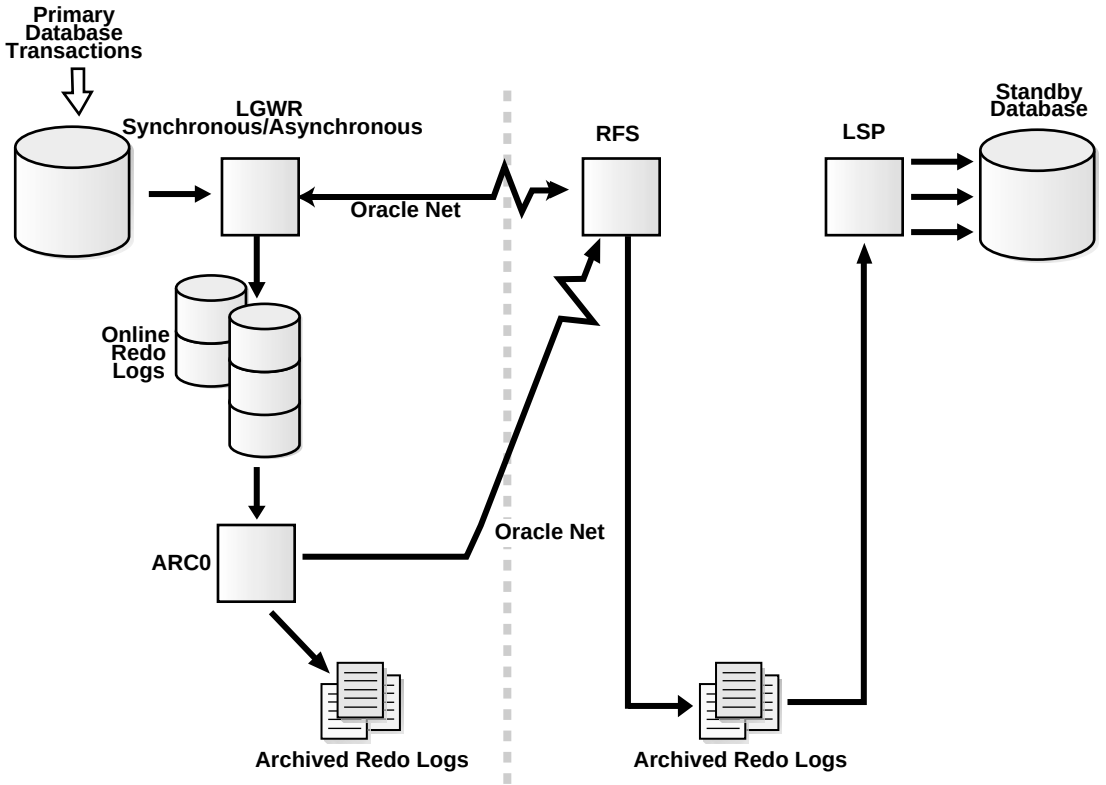


Figure 5–7 shows a Data Guard configuration with a local destination and a remote logical standby destination. In this configuration, the archiver on the primary system is archiving only to the local destination. Notice that the logwriter process is sending redo data to the standby system at the same time it is writing the data to the online redo logs. The RFS process receives the redo data and writes it to archived redo logs on the standby database. A log switch on the primary database triggers a log switch on the standby database, which causes the archiver process on the standby database to archive the redo logs on the standby database.

***Figure 5–7 Archiving to a Logical Standby Destination Using the Logwriter Process***



## 5.7 Setting the Data Protection Mode of a Data Guard Configuration

Each of the Data Guard data protection modes requires that *at least* one standby database in the configuration meet the minimum set of requirements listed in Table 5–2.

***Table 5–2 Requirements for Data Protection Modes***

| | Maximum Protection | Maximum Availability | Maximum Performance |
|---|---|---|---|
| Redo archival process | LGWR | LGWR | LGWR or ARCH |
| Network transmission mode | SYNC | SYNC | ASYNC when using LGWR process. Not applicable when using ARCH process |
| Disk write option | AFFIRM | AFFIRM | NOAFFIRM |
| Standby redo logs required? | Yes | Required for physical standby databases only | Required for physical standby databases using the LGWR process |
| Database type | Physical | Physical and logical | Physical and logical |

The standby database that is used to satisfy the minimum requirements for a given mode must be enabled and ready to receive redo data from the primary database before you can switch to that mode.

Before changing the data protection mode of your configuration, review the descriptions of the three data protection modes. Carefully consider the level of data protection that your business requires, and the performance and availability impact of operating in the mode that provides that level of protection.

---

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

---

After verifying that your Data Guard configuration meets the minimum requirements for the protection mode that you want to use, use the ALTER DATABASE SET STANDBY DATABASE TO MAXIMIZE statement to switch to that mode. The syntax for this statement is:

```
ALTER DATABASE SET STANDBY DATABASE TO MAXIMIZE {PROTECTION | AVAILABILITY | PERFORMANCE}
```

> **See Also:** Chapter 13 and *Oracle9i SQL Reference* for information about SQL statements

## 5.8 Log Transport Services Administration

The following sections describe how to control log transport services options using database initialization parameters.

### 5.8.1 Database Initialization Parameters

Although most initialization parameters on the primary and standby databases will be identical, some initialization parameters such as the CONTROL_FILES and DB_FILE_NAME_CONVERT parameters must differ. Only change parameter values when it is required for the functionality of the standby database or for filename conversions.

> **See Also:** Chapter 11 for a complete list of the initialization parameters that play a key role in the configuration of a standby database

#### 5.8.1.1 Setting Log Transport Parameters in the Initialization Parameter File

To set up log transport services, modify the database parameter initialization file before starting the database instance. When using a traditional text initialization parameter file, all parameters must be specified on one line.

---

**Note:** For the discussions in this section, examples are shown using a traditional text initialization parameter file so you can see the different ways you can specify parameters and changes.

---

Example 5–3 shows how to specify a parameter with a single attribute on one line.

***Example 5–3 Specifying a Single Attribute on One Line***

```
LOG_ARCHIVE_DEST_1='LOCATION=/disk1/oracle/oradata/payroll'
```

Example 5–4 shows how to specify a parameter with multiple attributes on one line.

***Example 5–4 Specifying Multiple Attributes on One Line***

```
LOG_ARCHIVE_DEST_1='LOCATION=/disk1/oracle/oradata/payroll LGWR'
```

For the majority of the log transport services initialization parameters, specifying a new value completely replaces any previously specified value.

#### 5.8.1.2 Setting Log Transport Parameters at Runtime Using SQL Statements

At runtime, the LOG_ARCHIVE_DEST_*n* initialization parameter can be changed using ALTER SYSTEM SET statements. You can specify the attributes in one or more strings in one statement. Any changes you make using the ALTER SYSTEM SET statements with the SCOPE=MEMORY clause are not persistently changed.

Example 5–5 shows how to specify a parameter with a single attribute on one line.

***Example 5–5 Specifying a Single Attribute on One Line***

```
ALTER SYSTEM SET LOG_ARCHIVE_DEST_1='LOCATION=/disk1/oracle/oradata/payroll';
```

For the majority of the log transport services initialization parameters, specifying a new value completely replaces any previously specified value.

### 5.8.2 Preparing Initialization Parameters for Role Transitions

Before performing a switchover or failover operation, you must configure certain initialization parameters on both the primary and standby databases so that your Data Guard configuration operates properly after a role transition.

During the creation process, you configured:

- Initialization parameters on the primary database to transmit redo data to the standby database
- Initialization parameters on the standby database to prepare the standby database to receive redo data
- A network address on the primary system for the standby database

- A network address on the standby system for the primary database

Now, you must set up the primary database to operate in the standby role and the standby database to operate in the primary role.

The following sections discuss only those initialization parameters that affect the log transport services and log apply services. The discussions do not mention any other parameters that you set during the creation of the primary and standby databases. Any parameter not mentioned in the following sections either must stay the same (such as the DATABASE_NAME parameter) or, if possible and necessary, can be modified to meet your requirements (such as the LOCK_NAME_SPACE or SHARED_POOL parameters).

#### 5.8.2.1 Primary Database Initialization Parameters

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

Example 5–6 shows the primary role initialization parameters that you maintain on the primary database.

***Example 5–6 Primary Database: Primary Role Initialization Parameters***

```
LOG_ARCHIVE_DEST_1='LOCATION=/disk1/oracle/oradata/payroll/'
LOG_ARCHIVE_DEST_2='SERVICE=sales1'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_STATE_2=ENABLE
LOG_ARCHIVE_FORMAT=%d_%t_%s.arc
REMOTE_ARCHIVE_ENABLE=SEND
```

These parameters control how log transport services send redo data to the standby system and the archiving of redo data on the local file system. The last parameter, REMOTE_ARCHIVE_ENABLE=SEND, allows the primary database to send redo data to the standby database, but prevents the primary database from receiving redo data from another system.

Example 5–7 shows the additional standby role initialization parameters on the primary database. These parameters take effect when the primary database is transitioned to the standby role.

***Example 5–7 Primary Database: Standby Role Initialization Parameters***

```
FAL_SERVER=sales1
```

```
FAL_CLIENT=sales
DB_FILE_NAME_CONVERT=('/standby','/primary')
LOG_FILE_NAME_CONVERT=('/standby','/primary')
STANDBY_ARCHIVE_DEST=/disk1/oracle/oradata/payroll/
STANDBY_FILE_MANAGEMENT=AUTO
```

Specifying the initialization parameters shown in Example 5–7 sets up the primary database to resolve gaps and convert new data and log file path names from a new primary database and archives the incoming redo data when this database is in the standby role.

#### 5.8.2.2 Standby Database Initialization Parameters

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

Example 5–8 shows the standby role initialization parameters that you would maintain on the standby database.

***Example 5–8 Standby Database: Standby Role Initialization Parameters***

```
FAL_SERVER=sales
FAL_CLIENT=sales1
DB_FILE_NAME_CONVERT=("/primary","/standby")
LOG_FILE_NAME_CONVERT=("/primary","/standby")
STANDBY_ARCHIVE_DEST=/disk1/oracle/oradata/payroll/standby/arc
LOG_ARCHIVE_DEST_1='LOCATION=/disk1/oracle/oradata/payroll/standby/arc/'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_FORMAT=%d_%t_%s.arc
STANDBY_FILE_MANAGEMENT=AUTO
REMOTE_ARCHIVE_ENABLE=RECEIVE
```

These initialization parameters allow the standby database to:

- Resolve gaps and convert new data and log file path names from the primary database
- Receive and archive the incoming redo data from the primary database, but only while the database is running in the standby role

The last parameter allows the standby database to receive redo data from a primary database, but it prevents the standby database from sending redo data.

Example 5–9 shows the additional primary role parameters to be added to the standby database that take effect when the standby database is transitioned to the primary role.

***Example 5–9 Standby Database: Primary Role Initialization Parameters***

```
LOG_ARCHIVE_DEST_2='SERVICE=sales'
LOG_ARCHIVE_DEST_STATE_2=ENABLE
```

These additional parameters control how log transport services send redo data to a new standby system.

#### 5.8.2.3 Enabling Initialization Parameters During Role Transition

With the initialization parameters on both the primary and standby databases set as described in Section 5.8.2.1 and Section 5.8.2.2, the only parameter that needs to change after a role transition is the REMOTE_ARCHIVE_ENABLE parameter. Change this parameter on both the original primary database and the standby database that assumes the primary role.

On the original primary database (the new standby) set this parameter to allow the receipt of the redo from the new primary database. For example:

```
SQL> ALTER SYSTEM SET REMOTE_ARCHIVE_ENABLE=RECEIVE SCOPE=MEMORY;
```

On the new primary database (the former standby) set this initialization parameter to allow the sending of redo to the standby database.

```
SQL> ALTER SYSTEM SET REMOTE_ARCHIVE_ENABLE=SEND SCOPE=MEMORY;
```

Setting the initialization parameter using the SCOPE=MEMORY clause ensures that the two databases will revert back to their original settings when the role transition is reversed, and the databases resume their original roles. If you expect that these databases will be restarted at some point without performing a role transition, replace the SCOPE=MEMORY with SCOPE=BOTH. In this event, this initialization parameter will have to be reset manually again after a new role transition.

#### 5.8.2.4 Logical Standby Database Considerations

The parameter values shown in the examples in Section 5.8.2.1 and Section 5.8.2.2 are suitable for both a physical and a logical standby configuration. If you plan to perform role transitions between a primary database and a logical standby database, then you must set the archiving destinations differently on the logical standby database and eventually on the primary database (for when it transitions to the logical standby database role).

When you create the logical standby database, you should specify different directories for the initialization parameters shown in Example 5–10.

***Example 5–10 Logical Standby Database: Standby Role Initialization Parameters***

```
STANDBY_ARCHIVE_DEST=/disk1/oracle/oradata/payroll/standby/incoming
LOG_ARCHIVE_DEST_1='LOCATION=/disk1/oracle/oradata/payroll/standby/arc/'
```

These parameters will continue to function correctly when the standby database assumes the primary role.

When the primary database assumes a logical standby role after a role transition, you must also configure the local archiving parameters to transmit the incoming redo data to a different location (as you did with the logical standby database).

***Example 5–11 Primary Database: Standby Role Initialization Parameters***

```
STANDBY_ARCHIVE_DEST=/disk1/oracle/oradata/payroll/incoming
LOG_ARCHIVE_DEST_1='LOCATION=/disk1/oracle/oradata/payroll/arc/'
```

The parameter values shown in Example 5–11 ensure that the redo data stream coming from the primary database will be archived to a different location than where the archive logs generated by the database are located when it is in the logical standby role.

## 5.9 Monitoring Redo Log Archival Information

This section describes manual methods of monitoring redo log archival activity for the primary database.

> **See Also:** *Oracle9i Data Guard Broker* and the Data Guard Manager online help for more information about the Oracle9*i* Data Guard Manager graphical user interface that automates many of the tasks involved in monitoring a Data Guard environment

### Step 1 Determine the current redo log sequence numbers.

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

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

THREAD#   SEQUENCE#  ARC  STATUS
--------  ---------  ---  ------
```

```
       1        947  YES   ACTIVE
       1        948  NO    CURRENT
```

### Step 2 Determine the most recently archived redo log.

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

```
SQL> SELECT MAX(SEQUENCE#) FROM V$ARCHIVED_LOG;

MAX(SEQUENCE#)
--------------
          947
```

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

Enter the following query at the primary database to determine the most recently archived redo log file to each of the archive destinations:

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

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

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

### Step 4 Find out if logs have been received at a particular site.

You can issue a query at the primary database to find out if a log was not sent to a particular site. Each archive destination has an ID number associated with it. You can query the DEST_ID column of the V$ARCHIVE_DEST fixed view on the primary database to identify archive destination IDs.

Assume the current local archive destination is 1, and one of the remote standby archive destination IDs is 2. To identify which logs were not received by this standby destination, issue the following query:

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

```
  5> (SELECT SEQUENCE# FROM V$ARCHIVED_LOG WHERE DEST_ID=2 AND
  6> THREAD# = LOCAL.THREAD#);

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

**See Also:** Appendix A, "Troubleshooting the Standby Database" to learn more about monitoring the archiving status of the primary database

### Step 5 Trace the progression of archived redo logs on the standby site.

To see the progression of the archiving of redo logs to the standby site, set the LOG_ARCHIVE_TRACE parameter in the primary and standby initialization parameter files.

**See Also:** Section 6.7 for complete details and examples.

# 6

# Log Apply Services

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

- Introduction to Log Apply Services
- Applying Redo Data to Physical Standby Databases
- Applying Redo Data to Logical Standby Databases
- Managing Archive Gaps
- Monitoring Log Apply Services for Physical Standby Databases
- Monitoring Log Apply Services for Logical Standby Databases
- Setting Archive Tracing

## 6.1 Introduction to Log Apply Services

**Log apply services** automatically apply archived redo logs to maintain synchronization with the primary database and allow transactionally consistent access to the data. Archived redo data is not available for log apply services until a log switch occurs on the primary database.

The main difference between physical and logical standby databases is the manner in which log apply services apply the archived redo logs. For physical standby databases, log apply services maintain the standby database by performing managed recovery operations. For logical standby databases, log apply services maintain the standby database by executing SQL statements. The following list summarizes these operations:

- Managed recovery operations (physical standby databases only)

In this mode, log transport services transmit redo data to the standby site, and log apply services automatically apply the redo logs.

> **Caution:** You can also open a physical standby database for read-only operations to allow users to query the standby database for reporting purposes. However, while a standby database that is open for read-only access, it is not kept transactionally current with the primary database, resulting in prolonging a failover or switchover operation if one is required for disaster recovery. See Section 8.2, "Using a Standby Database That Is Open for Read-Only Access" for more information.

- SQL apply operations (logical standby databases only)

  Log apply services manage logical standby databases by executing SQL statements. Logical standby databases can be opened in read / write mode, but the target tables being maintained by the logical standby database are opened in read-only mode for reporting purposes. The SQL apply mode allows you to use the logical standby database for reporting activities even while SQL statements are being applied.

The sections in this chapter describe the managed recovery and SQL apply operations, and log apply services in more detail.

## 6.2 Applying Redo Data to Physical Standby Databases

The physical standby database uses several processes to automate archiving redo data and recovering redo logs on the standby database. On the standby database, log apply services use the following processes:

- Remote file server (RFS)

  The remote file server (RFS) process receives redo data from the primary database either in the form of archived redo logs or standby redo logs.

- Archiver (ARC*n*)

  If standby redo logs are being used, the ARC*n* process archives the standby redo logs that are to be applied by the managed recovery process (MRP).

- Managed recovery process (MRP)

  The managed recovery process (MRP) applies information from the archived redo logs to the standby database. When performing managed recovery

operations, log apply services automatically apply archived redo logs to maintain transactional synchronization with the primary database.

Log apply services can apply logs to a physical standby database when the database is performing recovery, but not when it is open for read-only operations). A physical standby database can be performing one of the following:

- Managed recovery operations
- Read-only operations

Table 6–1 summarizes the basic tasks for configuring and monitoring log apply services.

***Table 6–1 Task List: Configuring Log Apply Services for Physical Standby Databases***

| Step | Task | See ... |
|---|---|---|
| 1 | Start the standby instance and mount the standby database. | Section 6.2.1 |
| 2 | Enable managed recovery or read-only operations. | Section 6.2.2.1 or Section 8.2, respectively |
| 3 | If performing managed recovery operations, set initialization parameters to automatically resolve archive gaps. | Section 6.4 and the *Oracle9i Net Services Administrator's Guide* |
| 4 | Monitor log apply services. | Section 6.5 |

## 6.2.1 Starting the Physical Standby Instance

After all necessary parameter and network files are configured, you can start the standby instance. If the standby instance is not started and mounted, the standby database cannot receive redo data from the primary database.

To start the physical standby database instance, perform the following steps:

1. Start the physical standby instance without mounting the database:

```
SQL> STARTUP NOMOUNT;
```

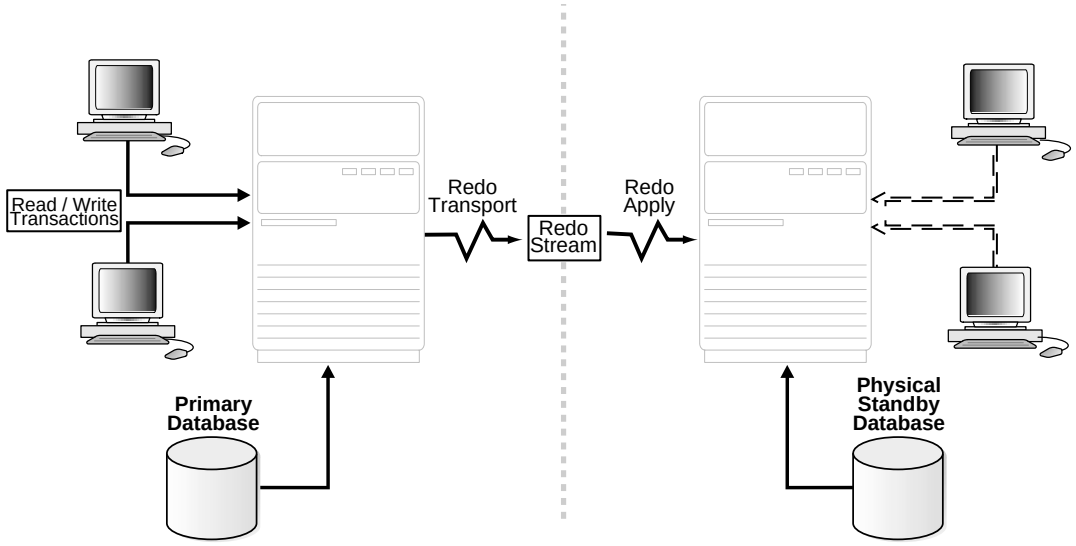
2. Mount the physical standby database. For example:

```
SQL> ALTER DATABASE MOUNT STANDBY DATABASE;
```

## 6.2.2 Starting Managed Recovery Operations

Log apply services keep the standby database synchronized with the primary database by automatically applying archived redo logs to the standby database, as shown in Figure 6–1.

***Figure 6–1 Automatic Updating of a Physical Standby Database***




### 6.2.2.1 Starting Log Apply Services

You can specify that log apply services run as a foreground session or as a background process.

- To start a foreground session, issue the SQL statement:

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

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

- To start a background process, you *must* use the `DISCONNECT` keyword on the SQL statement. For example:

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

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

- If you did not start log apply services as a detached server process, you can stop log apply services by the issuing the following SQL statement in another window:

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

**See Also:** Section 6.2.3 and Chapter 13

#### 6.2.2.2 Monitor the Recovery Process

You can query views to monitor log apply services as follows:

1. To verify that you have correctly initiated log apply services, query the V$MANAGED_STANDBY fixed view on the standby database. This view monitors the progress of a standby database in managed recovery mode. For example:

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

PROCESS STATUS       THREAD#    SEQUENCE#  BLOCK#     BLOCKS
------- ------------ ---------- ---------- ---------- ----------
MRP0    APPLYING_LOG 1          946        10         1001
```

   If you did not start a detached server process, you need to execute this query from another SQL session.

2. To monitor activity on the standby database, query the V$ARCHIVE_DEST_STATUS fixed view.

**See Also:** Section 6.5

### 6.2.3 Controlling Redo Apply Operations

Although this SQL ALTER DATABASE RECOVER MANAGED STANDBY DATABASE statement does not require any additional clauses, it provides many keywords to help you control the redo apply process.

**See Also:** Section 13.12 and *Oracle9i SQL Reference* for complete information about the SQL statement syntax

## 6.2.4 Datafile Management

To enable the automatic creation of new datafiles on a physical standby database when datafiles are created on the primary database, you must define the STANDBY_FILE_MANAGEMENT initialization parameter.

If the directory structures on the primary and standby databases are different, you must also set the DB_FILE_NAME_CONVERT initialization parameter to convert the filenames of one or more sets of datafiles on the primary database to filenames on the standby database.

### 6.2.4.1 Setting the STANDBY_FILE_MANAGEMENT Initialization Parameter

When you set the STANDBY_FILE_MANAGEMENT initialization parameter to AUTO, it automatically creates on the standby database any datafiles that were newly created on the primary database, using the same name that you specified on the primary database.

The STANDBY_FILE_MANAGEMENT initialization parameter works with the DB_FILE_NAME_CONVERT parameter to convert the datafile locations from the primary site to standby site.

### 6.2.4.2 Setting the DB_FILE_NAME_CONVERT Initialization Parameter

When a new datafile is added on the primary database, the same datafile is created on the standby database. The DB_FILE_NAME_CONVERT parameter is used to convert the datafile name on the primary database to a datafile name on the standby database. This parameter works the same if the STANDBY_FILE_MANAGEMENT initialization parameter is set to AUTO or MANUAL.

The DB_FILE_NAME_CONVERT initialization 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. For example:

```
DB_FILE_NAME_CONVERT= "/disk1/oracle/oradata/payroll/df1", \
"/disk1/oracle/oradata/payroll/standby/df1", \
"/disk1/oracle/oradata/payroll", "/disk1/oracle/oradata/payroll/standby/"
STANDBY_FILE_MANAGEMENT=AUTO
```

**Note:** When you specify pairs of files, be sure to specify the most restrictive path names before the least restrictive, as shown in the example.

#### 6.2.4.3 Restrictions on ALTER DATABASE Operations

You cannot rename the datafile on the standby site when the STANDBY_FILE_MANAGEMENT initialization parameter is set to AUTO. When you set the STANDBY_FILE_MANAGEMENT initialization parameter to AUTO, use of the following SQL statements is not allowed:

- ALTER DATABASE RENAME
- ALTER DATABASE ADD/DROP LOGFILE
- ALTER DATABASE ADD/DROP STANDBY LOGFILE MEMBER
- ALTER DATABASE CREATE DATAFILE AS

If you attempt to use any of these statements on the standby database, an error is returned. For example:

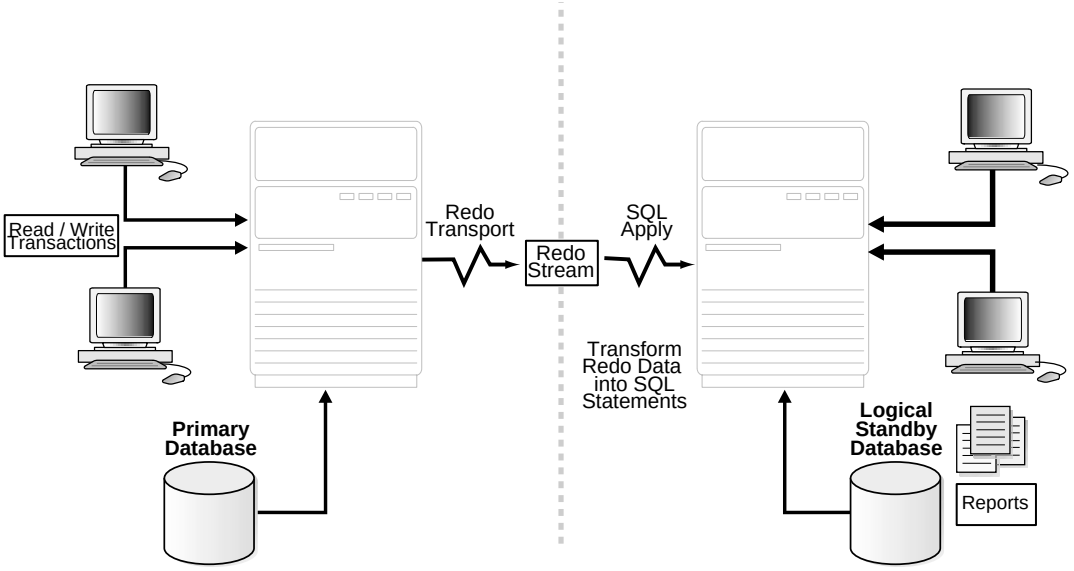
```
SQL> ALTER DATABASE RENAME FILE '/disk1/oracle/oradata/payroll/t_db2.log' to 'dummy';
alter database rename file '/disk1/oracle/oradata/payroll/t_db2.log' to 'dummy'
*
ERROR at line 1:
ORA-01511: error in renaming log/data files
ORA-01270: RENAME operation is not allowed if STANDBY_FILE_MANAGEMENT is auto
```

**See Also:** Section 8.4.1 to learn how to add datafiles to a database

## 6.3 Applying Redo Data to Logical Standby Databases

Log apply services convert the data from the redo logs 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. Figure 6–2 shows log apply services applying redo data to a logical standby database.

***Figure 6–2 Automatic Updating of a Logical Standby Database***




The logical standby database uses the following processes:

- Remote file server (RFS)

  The remote file server process receives redo data from the primary database. The RFS process communicates with the logical standby process (LSP) to coordinate and record which files arrived.

- Logical standby process (LSP)

  The logical standby process is the coordinator process for a set of processes that concurrently read, prepare, build, analyze, and apply completed SQL transactions from the archived redo logs. The LSP also maintains metadata in the database.

Table 6–2 summarizes the basic tasks for configuring log apply services.

***Table 6–2 Task List: Configuring Log Apply Services for Logical Standby Databases***

| Step | Task | See ... |
|---|---|---|
| 1 | Start log apply services. | Section 6.3.1 |
| 2 | Ensure that redo logs are being applied. | Section 6.3.2 |

*Table 6–2 Task List: Configuring Log Apply Services for Logical Standby Databases*

| Step | Task | See ... |
|---|---|---|
| 3 | Manage SQL apply operations. | Section 9.1 |

In addition to providing detailed information about the tasks presented in Table 6–2, the following sections also describe how to delay the application of archived redo logs.

## 6.3.1 Starting and Stopping Log Apply Services

To start log apply services, start the logical standby database, and then use the following statement. (Starting a logical standby database is done in the same manner as starting a primary database.)

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

To stop log apply services, use the following statement:

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

## 6.3.2 Ensuring That Redo Logs Are Being Applied

Redo logs are read and applied to a logical standby database when a log switch occurs, not as they arrive on the standby site. You can verify the status of archived redo log apply operations by querying the following views:

- V$LOGSTDBY

  Use this view to verify that the archived redo logs are being applied to the standby database. This view provides information about the processes that are reading redo data and applying archived redo logs to logical standby databases. For example, the following query shows typical output during the initialization phase:

```
SQL> COLUMN STATUS FORMAT A50
SQL> COLUMN TYPE FORMAT A12
SQL> SELECT TYPE, HIGH_SCN, STATUS FROM V$LOGSTDBY;
TYPE           HIGH_SCN STATUS
------------ ---------- --------------------------------------------------
COORDINATOR             ORA-16115: loading Log Miner dictionary data
READER                  ORA-16127: stalled waiting for additional transact
                        ions to be applied
BUILDER                 ORA-16117: processing
```

```
PREPARER                ORA-16116: no work available

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

- DBA_LOGSTDBY_PROGRESS

  Use this view for information about the progress of the log apply services. This view shows the state of the LSP and information about the SQL transactions that were executed on the logical standby database. For example:

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

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

  When the numbers in the APPLIED_SCN and NEWEST_SCN columns are equal (as shown in the query example), it means that all of the available data in the redo log was applied. These values can be compared to the values in the FIRST_CHANGE# column in the DBA_LOGSTDBY_LOG view to see how much log information has to be applied and how much remains.

  **See Also:** Chapter 9 for information about managing a logical standby database and Chapter 14 for more information about views that are used in a Data Guard environment

## 6.4 Managing Archive Gaps

Data Guard offers automatic archive redo log gap detection and resolution to handle network connectivity problems that might temporarily disconnect one or more standby databases from the primary database. Once properly configured, Data Guard requires no manual intervention by the DBA to detect and resolve such gaps.

The following sections describe gap detection and resolution.

## 6.4.1 What Is an Archive Gap?

An **archive gap** is a range of archived redo logs created whenever the standby system is unable to receive the next archived redo log generated by the primary database. For example, an archive gap occurs when the network becomes unavailable and automatic archiving from the primary database to the standby database stops. When the network is available again, automatic transmission of the redo data from the primary database to the failed standby database resumes.

The missing archived redo logs are the gap. The gap is automatically detected and resolved.

## 6.4.2 When Is an Archive Gap Discovered?

An archive gap can occur whenever the primary database archives a log, but the log is not archived to the standby site. Every minute, the primary database polls its standby databases to see if there is a gap in the sequence of archived redo logs. The polling between the primary and standby databases is sometimes referred to as a heartbeat. The primary database polls the standby databases serially.

## 6.4.3 Determining If an Archive Gap Exists on a Physical Standby Database

The following sections describe how to query the appropriate views to determine which logs are missing on the standby database.

### On a physical standby database

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

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

    THREAD#  LOW_SEQUENCE#  HIGH_SEQUENCE#
----------  -------------  --------------
          1              7              10
```

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

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

```
NAME
--------------------------------------------------------------------------------
/primary/thread1_dest/arcr_1_7.arc
/primary/thread1_dest/arcr_1_8.arc
/primary/thread1_dest/arcr_1_9.arc
```

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

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

After you register these logs on the physical standby database, you can restart managed recovery operations.

---

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

---

### On a logical standby database:

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

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

   THREAD#  SEQUENCE# FILE_NAME
```

```
---------- ---------- --------------------------------------------
         1          6 /disk1/oracle/dbs/log-1292880008_6.arc
         1         10 /disk1/oracle/dbs/log-1292880008_10.arc
```

Copy the missing logs to the logical standby system and register them using the ALTER DATABASE REGISTER LOGICAL LOGFILE statement on your logical standby database. For example:

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

After you register these logs on the logical standby database, you can restart log apply services.

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

## 6.4.4 How Is a Gap Resolved?

For both physical and logical standby databases, Data Guard performs gap detection and resolution automatically. No extra configuration settings are required. However, for physical standby databases, you can set initialization parameters so that log apply services also automatically resolve archive gaps as they occur on a physical standby database.

The following sections describe how to set initialization parameters to facilitate gap recovery for a physical standby database, and how gap recovery is handled on a logical standby database.

### On a physical standby database

You can set initialization parameters so that log apply services automatically identify and resolve archive gaps as they occur on a physical standby database.

Define the FAL_CLIENT and FAL_SERVER initialization parameters only for physical standby databases in the initialization parameter file:

| Parameter | Function | Syntax |
|---|---|---|
| FAL_CLIENT | This parameter specifies the network service name that the FAL server should use to connect to the standby database. | **Syntax**<br>FAL_CLIENT=*net_service_name*<br>**Example**<br>FAL_CLIENT=standby1_db |
| FAL_SERVER | This parameter specifies the network service name that the standby database should use to connect to the FAL server. | **Syntax**<br>FAL_SERVER=*net_service_name*<br>**Example**<br>FAL_SERVER=my_primary_db, my_standby_db |

The FAL server is a background Oracle process that services the incoming requests from the FAL client. In most cases, the FAL server is located on a primary database. However, it can be located on another standby database.

For log apply services to automatically identify and resolve archive gaps, you must:

1. On the standby system, use Oracle Net Manager to configure the listener. Use the TCP/IP protocol and statically register the standby database service with the listener using the service name. This service name will serve as the FAL client.
2. Use Oracle Net Manager to create a network service name that the standby database can use to connect to the FAL server. The network service name should resolve to a connect descriptor that uses the same protocol, host address, port, and service name that you specified when you configured the listener on the FAL server system, which is typically the primary system. If you are unsure what values to use for these parameters, use Oracle Net Manager to display the listener configuration on the FAL server system.
3. In the initialization parameter file of the standby database, assign the network service name that you created in step 1 to the FAL_CLIENT initialization parameter, and assign the network service name that you created in step 2 to the FAL_SERVER initialization parameter.
4. On the FAL server system, use Oracle Net Manager to create a network service name that the FAL server can use to connect to the standby database. The network service name should resolve to a connect descriptor that uses the same protocol, host address, port, and SID as the one in step 1.

Log apply services automatically detect, and the FAL server process running on the primary database attempts to resolve, any gaps that may exist when you enable managed recovery with the `ALTER DATABASE RECOVER MANAGED STANDBY DATABASE` statement.

> **See Also:** Section B.3 for a description of the manual steps and *Oracle9i Net Services Administrator's Guide* for information about Oracle Net

#### On a logical standby database

Gap recovery on a logical standby database is handled through the heartbeat mechanism. The important consideration here is that automatic gap recovery is contingent on the availability of the primary database. If the primary database is not available, as would be the case in a failover scenario, automatic gap recovery will not take place.

## 6.5 Monitoring Log Apply Services for Physical Standby Databases

To monitor the status of archived redo logs and obtain information on log apply services on a physical standby database, query the fixed views described in this section. You can also monitor the standby database using Data Guard Manager.

> **See Also:** Appendix A, "Troubleshooting the Standby Database"

This section contains the following topics:

- Accessing the V$MANAGED_STANDBY Fixed View
- Accessing the V$ARCHIVE_DEST_STATUS Fixed View
- Accessing the V$ARCHIVED_LOG Fixed View
- Accessing the V$LOG_HISTORY Fixed View
- Accessing the V$DATAGUARD_STATUS Fixed View

> **See Also:** Chapter 14 for complete reference information on the views named in the preceding list

### 6.5.1 Accessing the V$MANAGED_STANDBY Fixed View

Query the physical standby database to monitor log apply and log transport services activity at the standby site.

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

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

The previous query output shows that an RFS process has completed the archiving of redo log file sequence number 947. The output also shows a managed recovery operation that is actively applying archived redo log sequence number 946. The recovery operation is currently recovering block number 10 of the 72-block archived redo log.

### 6.5.2 Accessing the V$ARCHIVE_DEST_STATUS Fixed View

To quickly determine the level of synchronization for the standby database, issue the following query on the physical standby database:

```
SQL> SELECT ARCHIVED_THREAD#, ARCHIVED_SEQ#, APPLIED_THREAD#, APPLIED_SEQ#
  2> FROM V$ARCHIVE_DEST_STATUS;

ARCHIVED_THREAD# ARCHIVED_SEQ# APPLIED_THREAD# APPLIED_SEQ#
---------------- ------------- --------------- ------------
1                947           1               945
```

The previous query output shows that the standby database is two archived logs behind in applying the redo logs received from the primary database. This might indicate that a single recovery process is unable to keep up with the volume of archived redo logs being received. Using the PARALLEL option might be a solution.

### 6.5.3 Accessing the V$ARCHIVED_LOG Fixed View

The V$ARCHIVED_LOG fixed view on the physical standby database shows all the archived redo logs received from the primary database. This view is only useful after the standby site starts receiving logs, because before that time the view is populated by old archived log records generated from the primary control file. For example, you can execute the following SQL*Plus statement:

```
SQL> SELECT REGISTRAR, CREATOR, THREAD#, SEQUENCE#, FIRST_CHANGE#,
  2> NEXT_CHANGE# FROM V$ARCHIVED_LOG;

REGISTRAR CREATOR THREAD#    SEQUENCE#  FIRST_CHANGE# NEXT_CHANGE#
--------- ------- ---------- ---------- ------------- ------------
```

```
RFS       ARCH    1          945        74651         74739
RFS       ARCH    1          946        74739         74772
RFS       ARCH    1          947        74772         74774
```

The previous query output shows three archived redo logs received from the primary database.

**See Also:** V$ARCHIVED_LOG in Chapter 14

### 6.5.4 Accessing the V$LOG_HISTORY Fixed View

Query the V$LOG_HISTORY fixed view on the physical standby database to show all the archived redo logs that were applied:

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

THREAD#    SEQUENCE#  FIRST_CHANGE# NEXT_CHANGE#
---------- ---------- ------------- ------------
1          945        74651         74739
```

The previous query output shows that the most recently applied archived redo log was sequence number 945.

### 6.5.5 Accessing the V$DATAGUARD_STATUS Fixed View

The V$DATAGUARD_STATUS fixed view displays events that would typically be triggered by any message to the alert log or server process trace files.

The following example shows output from the V$DATAGUARD_STATUS view on a primary database:

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

MESSAGE
--------------------------------------------------------------------------------

ARC0: Archival started
ARC1: Archival started
Archivelog destination LOG_ARCHIVE_DEST_2 validated for no-data-loss
recovery
Creating archive destination LOG_ARCHIVE_DEST_2: 'dest2'
ARCH: Transmitting activation ID 0
LGWR: Completed archiving log 3 thread 1 sequence 11
Creating archive destination LOG_ARCHIVE_DEST_2: 'dest2'
```

```
LGWR: Transmitting activation ID 6877c1fe
LGWR: Beginning to archive log 4 thread 1 sequence 12
ARC0: Evaluating archive   log 3 thread 1 sequence 11
ARC0: Archive destination LOG_ARCHIVE_DEST_2: Previously completed
ARC0: Beginning to archive log 3 thread 1 sequence 11
Creating archive destination LOG_ARCHIVE_DEST_1:
'/oracle/arch/arch_1_11.arc'

ARC0: Completed archiving  log 3 thread 1 sequence 11
ARC1: Transmitting activation ID 6877c1fe

15 rows selected.
```

The following example shows the contents of the V$DATAGUARD_STATUS view on a physical standby database:

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

MESSAGE
--------------------------------------------------------------------------------

ARC0: Archival started
ARC1: Archival started
RFS: Successfully opened standby logfile 6: '/oracle/dbs/sorl2.log'

ARC1: Evaluating archive   log 6 thread 1 sequence 11
ARC1: Beginning to archive log 6 thread 1 sequence 11
Creating archive destination LOG_ARCHIVE_DEST_1:
'/oracle/arch/arch_1_11.arc'

ARC1: Completed archiving  log 6 thread 1 sequence 11
RFS: Successfully opened standby logfile 5: '/oracle/dbs/sorl1.log'

Attempt to start background Managed Standby Recovery process
Media Recovery Log /oracle/arch/arch_1_9.arc

10 rows selected.
```

**See Also:** V$DATAGUARD_STATUS in Chapter 14

## 6.6 Monitoring Log Apply Services for Logical Standby Databases

To monitor the status of archived redo logs and obtain information on log apply services on a logical standby database, query the fixed views described in this section. You can also monitor the standby database using Data Guard Manager.

**See Also:** Appendix A, "Troubleshooting the Standby Database"

This section contains the following topics:

- Accessing the DBA_LOGSTDBY_EVENTS View
- Accessing the DBA_LOGSTDBY_LOG View
- Accessing the DBA_LOGSTDBY_PROGRESS View
- Accessing the V$LOGSTDBY Fixed View
- Accessing the V$LOGSTDBY_STATS Fixed View

### 6.6.1 Accessing the DBA_LOGSTDBY_EVENTS View

If log apply services should stop unexpectedly, the reason for the problem is shown in this view.

---

**Note:** Errors that cause SQL apply operations to stop are always recorded in the events table (unless there is insufficient space in the system tablespace). These events are always put into the ALERT.LOG file as well, with the phrase '`LOGSTDBY event`' included in the text. When querying the view, select the columns in order by `EVENT_TIME`, `COMMIT_SCN`, and `CURRENT_SCN`. This ordering ensures that a shutdown failure appears last in the view.

---

The view also contains other information, such as which DDL statements were applied and which were skipped. For example:

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

SQL> COLUMN STATUS FORMAT A60
SQL> SELECT EVENT_TIME, STATUS, EVENT FROM DBA_LOGSTDBY_EVENTS
  2  ORDER BY EVENT_TIME, COMMIT_SCN;
```

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

--------------------------------------------------------------------------------
23-JUL-02 18:20:12 ORA-16111: log mining and apply setting up
23-JUL-02 18:20:12 ORA-16128: User initiated shut down successfully completed
23-JUL-02 18:20:12 ORA-16112: log mining and apply stopping
23-JUL-02 18:20:23 ORA-16111: log mining and apply setting up
23-JUL-02 18:55:12 ORA-16128: User initiated shut down successfully completed
23-JUL-02 18:57:09 ORA-16111: log mining and apply setting up
23-JUL-02 20:21:47 ORA-16204: DDL successfully applied
create table mytable (one number, two varchar(30))
23-JUL-02 20:22:55 ORA-16205: DDL skipped due to skip setting create database
link mydblink

8 rows selected.
```

This query shows that log apply services were started and stopped a few times. It also shows what DDL was applied and skipped. If log apply services had stopped, the last record in the query would have shown the cause of the problem.

## 6.6.2 Accessing the DBA_LOGSTDBY_LOG View

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

For example:

```
SQL> SELECT FILE_NAME, SEQUENCE#, FIRST_CHANGE#, NEXT_CHANGE#,
  2> TIMESTAMP, DICT_BEGIN, DICT_END, THREAD# FROM DBA_LOGSTDBY_LOG
  3> ORDER BY SEQUENCE#;

FILE_NAME                SEQ# FIRST_CHANGE# NEXT_CHANGE# TIMESTAM BEG END THR#
------------------------ ---- ------------- ------------ -------- --- --- ----
/oracle/dbs/hq_nyc_2.log  2           101579       101588 11:02:58 NO  NO  1
/oracle/dbs/hq_nyc_3.log  3           101588       142065 11:02:02 NO  NO  1
/oracle/dbs/hq_nyc_4.log  4           142065       142307 11:02:10 NO  NO  1
/oracle/dbs/hq_nyc_5.log  5           142307       142739 11:02:48 YES YES 1
/oracle/dbs/hq_nyc_6.log  6           142739       143973 12:02:10 NO  NO  1
/oracle/dbs/hq_nyc_7.log  7           143973       144042 01:02:11 NO  NO  1
/oracle/dbs/hq_nyc_8.log  8           144042       144051 01:02:01 NO  NO  1
/oracle/dbs/hq_nyc_9.log  9           144051       144054 01:02:16 NO  NO  1
```

```
/oracle/dbs/hq_nyc_10.log 10          144054       144057 01:02:21 NO  NO  1
/oracle/dbs/hq_nyc_11.log 11          144057       144060 01:02:26 NO  NO  1
/oracle/dbs/hq_nyc_12.log 12          144060       144089 01:02:30 NO  NO  1
/oracle/dbs/hq_nyc_13.log 13          144089       144147 01:02:41 NO  NO  1
```

The output from this query shows that a LogMiner dictionary build starts at log file sequence number 5. The most recent archive log file is sequence number 13 and it was received at the logical standby database at 01:02:41.

### 6.6.3 Accessing the DBA_LOGSTDBY_PROGRESS View

To quickly determine if all log file information was applied, issue the following query on the logical standby database:

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

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

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

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

SQL> SELECT L.SEQUENCE#, L.FIRST_TIME,
  2    (CASE WHEN L.NEXT_CHANGE# < P.READ_SCN THEN 'YES'
  3          WHEN L.FIRST_CHANGE# < P.APPLIED_SCN THEN 'CURRENT'
  4          ELSE 'NO' END) APPLIED
  5  FROM DBA_LOGSTDBY_LOG L, DBA_LOGSTDBY_PROGRESS P
  6  ORDER BY SEQUENCE#;

 SEQUENCE# FIRST_TIME         APPLIED
---------- ----------------- -------
        24 23-JUL-02 18:19:05 YES
        25 23-JUL-02 18:19:48 YES
        26 23-JUL-02 18:19:51 YES
        27 23-JUL-02 18:19:54 YES
        28 23-JUL-02 18:19:59 YES
        29 23-JUL-02 18:20:03 YES
        30 23-JUL-02 18:20:13 YES
        31 23-JUL-02 18:20:18 YES
```

```
        32 23-JUL-02 18:20:21 YES
        33 23-JUL-02 18:32:11 YES
        34 23-JUL-02 18:32:19 CURRENT
        35 23-JUL-02 19:13:20 CURRENT
        36 23-JUL-02 19:13:43 CURRENT
        37 23-JUL-02 19:13:46 CURRENT
        38 23-JUL-02 19:13:50 CURRENT
        39 23-JUL-02 19:13:54 CURRENT
        40 23-JUL-02 19:14:01 CURRENT
        41 23-JUL-02 19:15:11 NO
        42 23-JUL-02 19:15:54 NO

19 rows selected.
```

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

## 6.6.4 Accessing the V$LOGSTDBY Fixed View

To inspect the process activity for SQL apply operations, query the V$LOGSTDBY fixed view on the logical standby database. For example:

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

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

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

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

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

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

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

11 rows selected.
```

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

### 6.6.5 Accessing the V$LOGSTDBY_STATS Fixed View

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

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

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

This query shows how long SQL apply operations have been running and how many transactions have been applied in that time. It also shows how many transactions are available to be applied, indicating that more work is necessary.

# 6.7 Setting Archive Tracing

To see the progression of the archiving of redo logs to the standby site, set the LOG_ARCHIVE_TRACE parameter in the primary and standby initialization parameter files. When you set the LOG_ARCHIVE_TRACE parameter, it causes the Oracle database server to write an audit trail to a trace file as follows:

- On the primary database

  This causes the Oracle database server to write an audit trail of archiving process activity (ARC*n* and foreground processes) on the primary database in a trace file whose filename is specified in the USER_DUMP_DEST initialization parameter.

- On the standby database

  This causes the Oracle database server to write an audit trail of the RFS process and the ARC*n* process activity relating to archived redo logs on the standby database in a trace file whose filename is specified in the USER_DUMP_DEST initialization parameter.

## 6.7.1 Determining the Location of the Trace Files

The trace files for a database are located in the directory specified by the USER_DUMP_DEST parameter in the initialization parameter file. Connect to the primary and standby instances using SQL*Plus, and issue a SHOW statement to determine the location, for example:

```
SQL> SHOW PARAMETER user_dump_dest
NAME                                 TYPE    VALUE
------------------------------------ ------- ------------------------------
user_dump_dest                       string  ?/rdbms/log
```

## 6.7.2 Setting the Log Trace Parameter

The format for the archiving trace parameter is as follows, where *trace_level* is an integer:

```
LOG_ARCHIVE_TRACE=trace_level
```

To enable, disable, or modify the LOG_ARCHIVE_TRACE parameter in a primary database, do one of the following:

- Shut down the primary database, modify the initialization parameter file, and restart the database.

- Issue an ALTER SYSTEM SET LOG_ARCHIVE_TRACE=*trace_level* statement while the database is open or mounted.

To enable, disable, or modify the LOG_ARCHIVE_TRACE parameter for a physical standby database that is performing read-only or managed recovery operations, issue a SQL statement similar to the following:

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_TRACE=15;
```

In the previous example, setting the LOG_ARCHIVE_TRACE parameter to a value of 15 sets trace levels 1, 2, 4, and 8 as described in Section 6.7.3.

Issue the ALTER SYSTEM statement from a different standby session so that it affects trace output generated by the remote file service (RFS) and ARC*n* processes when the next archived log is received from the primary database. For example, enter:

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_TRACE=32;
```

## 6.7.3 Choosing an Integer Value

The integer values for the LOG_ARCHIVE_TRACE parameter represent levels of tracing data. In general, the higher the level, the more detailed the information. The following integer levels are available:

| Level | Meaning |
|---|---|
| 0 | Disables archived redo log tracing (default setting) |
| 1 | Tracks archiving of redo log file |
| 2 | Tracks archival status per archived redo log destination |
| 4 | Tracks archival operational phase |
| 8 | Tracks archived redo log destination activity |
| 16 | Tracks detailed archived redo log destination activity |
| 32 | Tracks archived redo log destination parameter modifications |
| 64 | Tracks ARC*n* process state activity |
| 128 | Tracks FAL server process activity |
| 256 | Supported in a future release |
| 512 | Tracks asynchronous LGWR activity |
| 1024 | Tracks the RFS physical client |

| Level | Meaning |
|---|---|
| 2048 | Tracks the ARC*n* or RFS heartbeat |

You can combine tracing levels by setting the value of the LOG_ARCHIVE_TRACE parameter to the sum of the individual levels. For example, setting the parameter to 6 generates level 2 and level 4 trace output.

The following are examples of the ARC0 trace data generated on the primary site by the archiving of redo log 387 to two different destinations: the service standby1 and the local directory /oracle/dbs.

> **Note:** The level numbers do not appear in the actual trace output; they are shown here for clarification only.

```
Level   Corresponding entry content (sample)
-----   -------------------------------
( 1)    ARC0: Begin archiving log# 1 seq# 387 thrd# 1
( 4)    ARC0: VALIDATE
( 4)    ARC0: PREPARE
( 4)    ARC0: INITIALIZE
( 4)    ARC0: SPOOL
( 8)    ARC0: Creating archive destination 2 : 'standby1'
(16)    ARC0:  Issuing standby Create archive destination at 'standby1'
( 8)    ARC0: Creating archive destination 1 : '/oracle/dbs/d1arc1_387.log'
(16)    ARC0:  Archiving block 1 count 1 to : 'standby1'
(16)    ARC0:  Issuing standby Archive of block 1 count 1 to 'standby1'
(16)    ARC0:  Archiving block 1 count 1 to :  '/oracle/dbs/d1arc1_387.log'
( 8)    ARC0: Closing archive destination 2  : standby1
(16)    ARC0:  Issuing standby Close archive destination at 'standby1'
( 8)    ARC0: Closing archive destination 1  :  /oracle/dbs/d1arc1_387.log
( 4)    ARC0: FINISH
( 2)    ARC0: Archival success destination 2 : 'standby1'
( 2)    ARC0: Archival success destination 1 : '/oracle/dbs/d1arc1_387.log'
( 4)    ARC0: COMPLETE, all destinations archived
(16)    ARC0: ArchivedLog entry added: /oracle/dbs/d1arc1_387.log
(16)    ARC0: ArchivedLog entry added: standby1
( 4)    ARC0: ARCHIVED
( 1)    ARC0: Completed archiving log# 1 seq# 387 thrd# 1

(32)  Propagating archive 0 destination version 0 to version 2
         Propagating archive 0 state version 0 to version 2
```

```
         Propagating archive 1 destination version 0 to version 2
         Propagating archive 1 state version 0 to version 2
         Propagating archive 2 destination version 0 to version 1
         Propagating archive 2 state version 0 to version 1
         Propagating archive 3 destination version 0 to version 1
         Propagating archive 3 state version 0 to version 1
         Propagating archive 4 destination version 0 to version 1
         Propagating archive 4 state version 0 to version 1

(64) ARCH: changing ARC0 KCRRNOARCH->KCRRSCHED
       ARCH: STARTING ARCH PROCESSES
       ARCH: changing ARC0 KCRRSCHED->KCRRSTART
       ARCH: invoking ARC0
       ARC0: changing ARC0 KCRRSTART->KCRRACTIVE
       ARCH: Initializing ARC0
       ARCH: ARC0 invoked
       ARCH: STARTING ARCH PROCESSES COMPLETE
       ARC0 started with pid=8
       ARC0: Archival started
```

The following is the trace data generated by the RFS process on the standby site as it receives archived log 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
```

# 7

# Role Management

A Data Guard configuration consists of one database that functions in a primary role and one or more databases that function in a standby role. Typically, the role of each database does not change. However, if the primary database becomes unavailable or if hardware or software maintenance operations must be performed, you might need to change the role of one or more databases in the configuration.

The number, location, and type (physical or logical) of standby databases in the Data Guard configuration and the way in which changes to the primary database are propagated to each standby database determine, in advance, the role management options available to you in response to a planned or unplanned primary database outage.

This chapter describes the Data Guard role management services and operations that allow you to change and manage the roles of the databases in a Data Guard configuration. It contains the following topics:



## 7.1 Introduction to Role Transitions

A database operates in one of the following mutually exclusive roles: **primary** or **standby**. Data Guard allows you to change these roles dynamically by issuing the SQL commands described in this chapter, or by using Data Guard Manager, or by using the Oracle Data Guard broker command-line interface as described in the *Oracle9i Data Guard Broker* guide.

Oracle Data Guard supports two role transition operations. The first operation, a **switchover,** is a reversible role transition between the primary database and one of

its standby databases. The second operation, a **failover,** transitions a standby database to the primary role in response to a failure of the primary database.

Each of these operations are described in more detail in Section 7.1.2 and Section 7.1.3. Section 7.1.1 helps you choose the role transition operation that best minimizes downtime and risk of data loss.

---

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

---

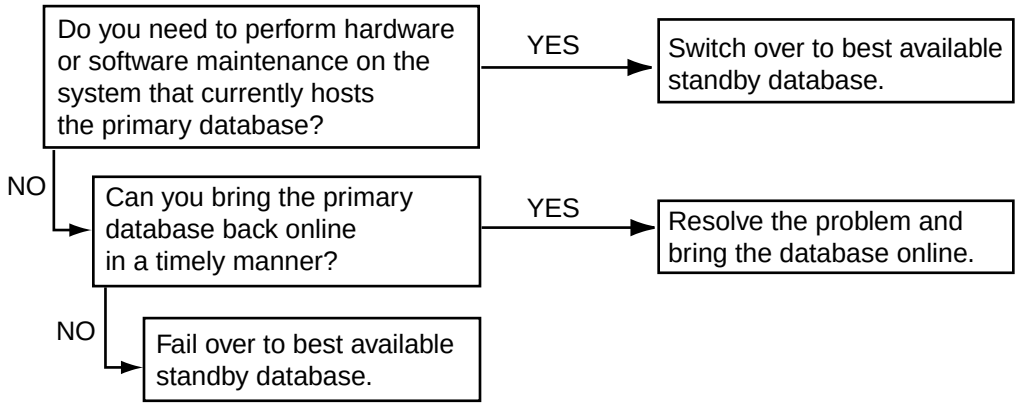
### 7.1.1 Which Role Transition to Use

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

- The state of the primary database just before the transition
- The state of the standby database selected for the role transition at the time of the transition
- If the selected standby database was configured as a physical standby database or a logical standby database
- If the role transition is a switchover or a failover

The goal is to perform the role transition as quickly as possible with little or no data loss. The decision tree presented in Figure 7–1 can help you choose the role transition operation that best minimizes downtime and risk of data loss.

***Figure 7–1 Role Transition Decision Tree***




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

If a role transition must be performed and the configuration contains physical standby databases, Oracle recommends that you perform the role transition using the best available physical standby database. Role transitions involving a logical standby database:

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

> **See Also:** Section 10.1 for information about how to choose the best available physical or logical standby database.

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

- For switchover operations, refer to Section 7.1.2
- For failover operations, refer to Section 7.1.3

## 7.1.2 Switchover Operations

During a switchover operation, there is no data loss, and the old primary database remains in the configuration as a standby database. A switchover is typically used to reduce primary database downtime during planned outages, such as operating system or hardware upgrades. A switchover operation takes place in two phases. In the first phase, the existing primary database is transitioned to a standby role. In the second phase, a standby database is transitioned to the primary role.

Figure 7–2 shows a two-site Data Guard configuration before the roles of the databases are switched. The primary database is in San Francisco, and the standby database is in Boston.

***Figure 7–2 Data Guard Configuration Before a Switchover Operation***

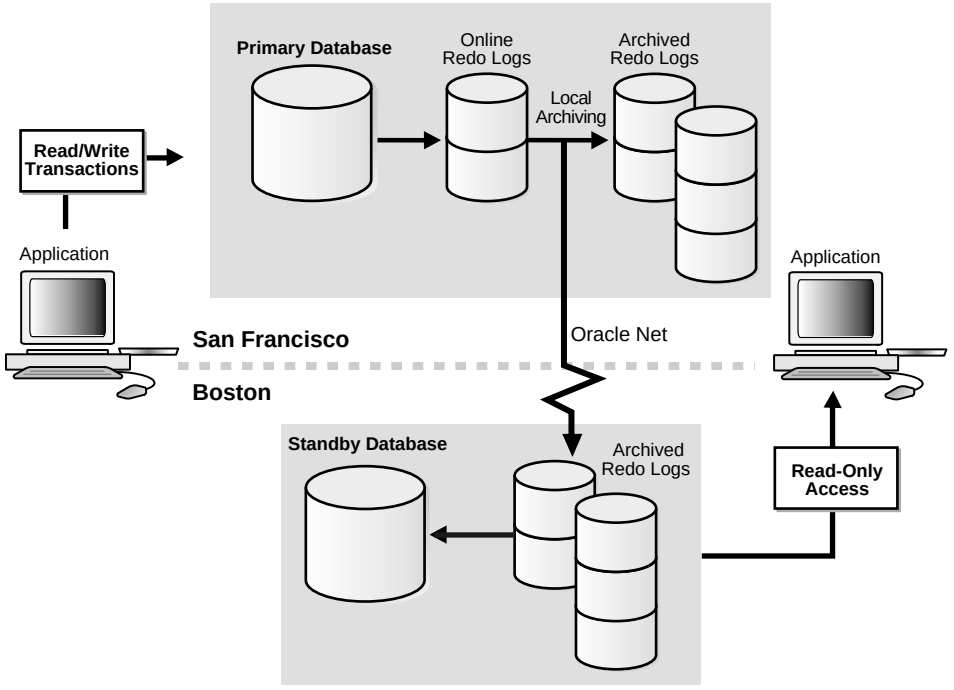


Figure 7–3 shows the Data Guard environment after the original primary database was switched over to a standby database, but before the original standby database has become the new primary database. At this stage, the Data Guard configuration temporarily has two standby databases.

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

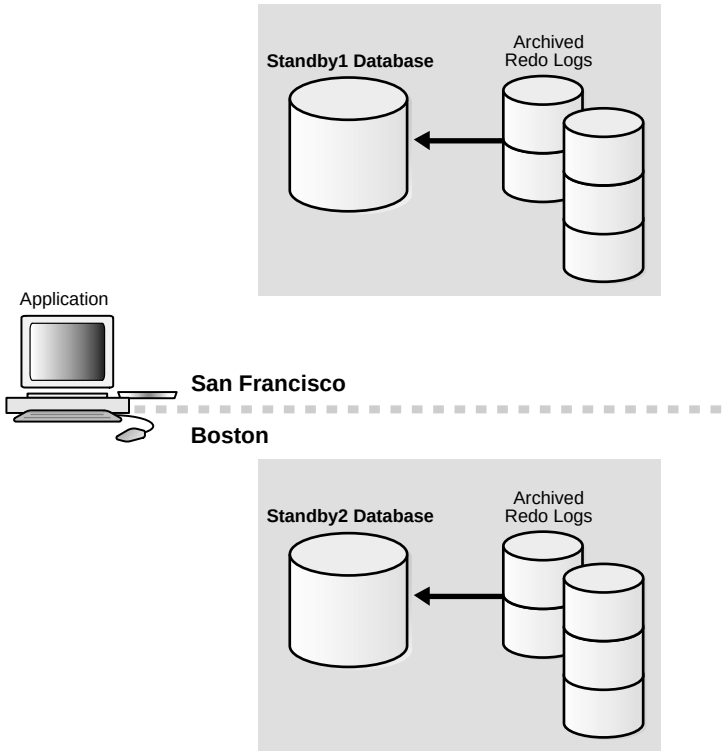
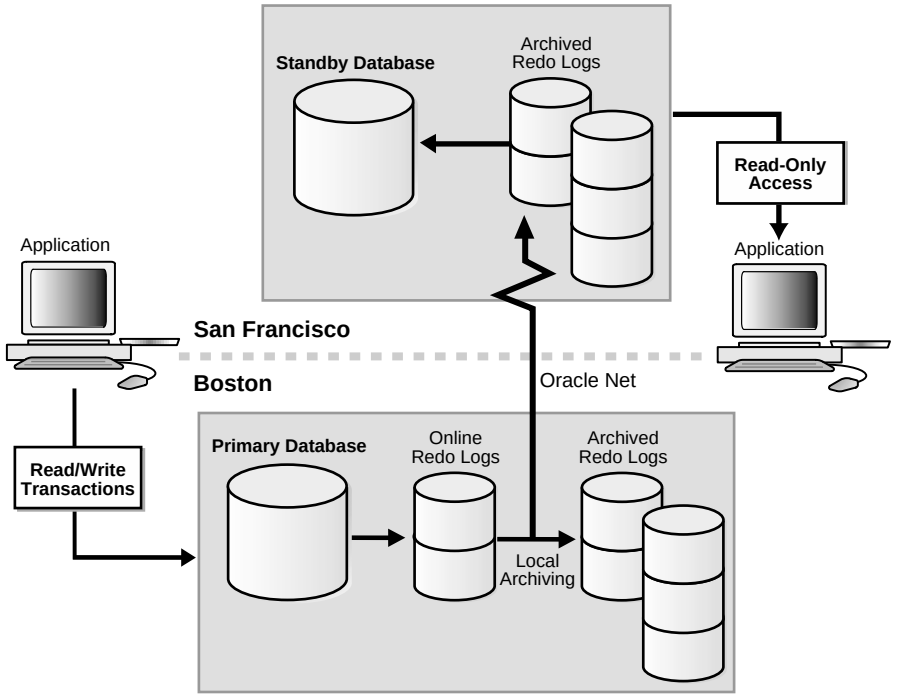


Figure 7–4 shows the Data Guard environment after a switchover took place. The original standby database became the new primary database. The primary database is now in Boston, and the standby database is now in San Francisco.

***Figure 7–4 Data Guard Environment After Switchover***




### 7.1.2.1 Preparing for a Switchover

Although switchover operations can be performed between the primary database and either a logical or a physical standby database in the Data Guard configuration, a physical standby database is preferred (as described in Section 7.1.1). To minimize downtime, carefully plan each switchover operation so that the primary and standby databases involved have as small a transactional lag as possible.

Before starting a switchover operation:

- Identify the initialization parameters that you must change to complete the role transition.

---

**Note:** If you do not use the Data Guard broker, you must define the LOG_ARCHIVE_DEST_*n* and LOG_ARCHIVE_DEST_STATE_*n* parameters on all standby sites so that when a switchover or failover operation occurs, all of the standby sites continue to receive logs from the *new* primary database. Configurations that you set up with the Data Guard broker command-line interface or Data Guard Manager handle the LOG_ARCHIVE_DEST_*n* and LOG_ARCHIVE_DEST_STATE_*n* definitions automatically, including defining the LOG_ARCHIVE_DEST_*n* parameters to point back to the primary database and all the other standby databases.

---

Be sure you refer to Section 5.8.2, which describes how to configure initialization parameters on both the primary and standby databases so that your Data Guard configuration operates properly after a role transition.

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

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

- Verify that there are no active users connected to the databases.

- Verify that all but one primary instance and one standby instance in a Real Application Clusters configuration are shut down.

  For a Real Application Clusters database, only one primary instance and one standby instance can perform the switchover operation. Shut down all other instances before the switchover operation.

- For switchover operations involving a physical standby database, the primary database instance is open and the standby database instance is mounted.

  The standby database that you plan to transition to the primary role must be mounted before you begin the switchover operation. Ideally, the physical standby database will also be actively recovering archived redo logs when the database roles are switched. If the physical standby database is open for read-only access, the switchover operation still will take place, but will require additional time.

  **See Also:** Section 6.2.2 for more information on managed recovery mode

- For switchover operations involving a logical standby database, both the primary and standby database instances are open.
- Place the standby database that will become the new primary database in ARCHIVELOG mode.
- Remove any redo data application delay in effect on the standby database.

> **Note:** Because the primary and standby database releases must be the same at all times, do not use a switchover operation to perform a rolling upgrade of Oracle database software. However, it might be possible to use a switchover operation to perform a rolling upgrade of system hardware.

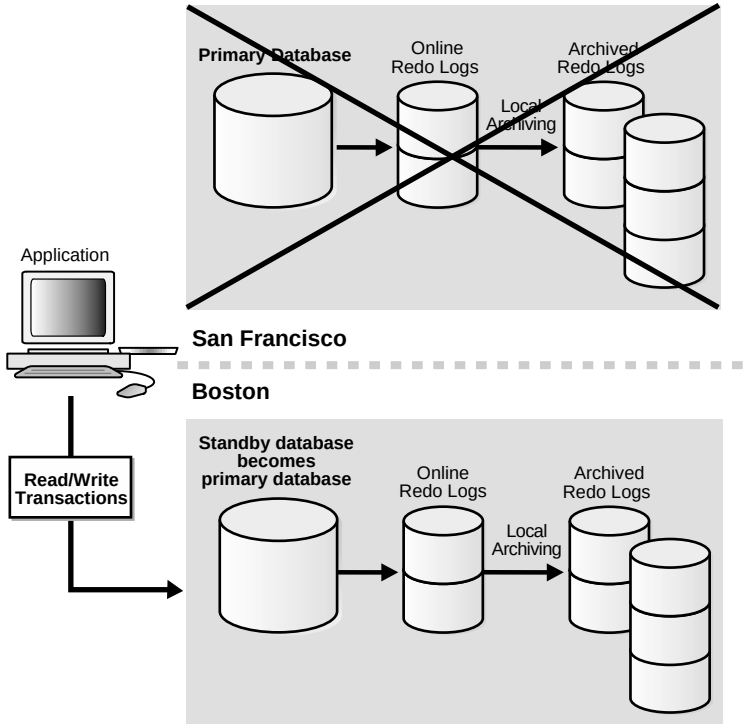
For switchover operations involving a physical standby database, refer to Section 7.2.1. For switchover operations involving a logical standby database, refer to Section 7.3.1. If you configured your environment using the Oracle Data Guard Broker distributed management framework, refer instead to the *Oracle9i Data Guard Broker* guide for information about how to use the Oracle Data Guard Manager Switchover Wizard to automate the switchover process.

### 7.1.3 Failover Operations

During a failover operation, a standby database transitions to the primary role and the old primary database is rendered unable to participate in the configuration. Depending on the protection mode under which the old primary database was operating before the failover, there might be little or no data loss during a failover. A failover is typically used only when a primary database becomes unavailable and there is no possibility of restoring it to service within a reasonable amount of time. The specific actions performed during a failover vary based on if a logical or a physical standby database is involved in the failover operation, the state of the configuration at the time of the failover, and on the specific SQL commands used to initiate the failover.

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

***Figure 7–5 Failover to a Standby Database***




**Note:** After performing a failover operation, you can optionally restore the original state of the Data Guard configuration by performing the following steps:

1. Re-create the failed primary database as a new standby database using a copy of the new primary database.
2. Add the database to the configuration as a new standby database.
3. Perform a switchover to transition the database to the primary role and restore the configuration to its original pre-failure state.

#### 7.1.3.1 Preparing for a Failover

In general, before performing a failover operation, you should transfer as much of the available and unapplied primary database redo data as possible to the standby database by following the steps described in this section.

Before starting a failover operation:

- Identify the parameters that must be changed to complete the role transition.

  **Note:** If you do not use the Data Guard broker, you must define the LOG_ARCHIVE_DEST_*n* and LOG_ARCHIVE_DEST_STATE_*n* parameters on all standby sites so that when a switchover or failover operation occurs, all of the standby sites continue to receive logs from the *new* primary database. Configurations that you set up with the Data Guard command-line interface or Data Guard Manager handle the LOG_ARCHIVE_DEST_*n* and LOG_ARCHIVE_DEST_STATE_*n* definitions automatically, including defining the LOG_ARCHIVE_DEST_*n* parameters to point back to the primary database and all the other standby databases.

  **See Also:** Section 5.8.2 provides sample primary and standby initialization parameter files.

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

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

- Verify that all but one standby instance in a Real Application Clusters configuration are shut down.

  For a Real Application Clusters database, only one standby instance can be active during the failover operation. Shut down all other instances before the failover operation.

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

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

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

---

**Note:** Do not fail over a primary database to a physical standby database to test whether or not the standby database is being updated correctly. Instead, open the standby database in read-only mode and query the database to ensure that updates made to the primary database were propagated to the standby database.

---

If you are performing a failover operation involving a physical standby database, refer to Section 7.2.2. If you are performing a failover operation involving a logical standby database, refer to Section 7.3.2.

## 7.2 Role Transitions Involving Physical Standby Databases

This section describes how to perform switchovers and failovers involving a physical standby database and recovering from an unsuccessful switchover.

### 7.2.1 Switchover Operations Involving a Physical Standby Database

This section describes how to perform a switchover operation that changes roles between a primary database and a physical standby database. Always initiate the switchover operation on the primary database and complete it on the physical standby database. The following steps describe how to perform the switchover operation.

#### On the current primary database

#### Step 1 Verify that it is possible to perform a switchover operation.

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

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

The TO STANDBY value in the SWITCHOVER_STATUS column indicates that it is possible to switch the primary database to the standby role. If the TO STANDBY value is not displayed, then verify that the Data Guard configuration is functioning correctly (for example, verify that all LOG_ARCHIVE_DEST_*n* parameter values are specified correctly).

> **See Also:** Chapter 14 for information about other valid values for the SWITCHOVER_STATUS column of the V$DATABASE view

### Step 2 Initiate the switchover operation on the primary database.

To transition the current primary database to a physical standby database role, use the following SQL statement on the primary database:

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

After this statement completes, the primary database is converted into a standby database. The current control file is backed up to the current SQL session trace file before the switchover operation. This makes it possible to reconstruct a current control file, if necessary.

### Step 3 Shut down and restart the former primary instance.

Shut down the former primary instance and restart it without mounting the database:

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

Mount the database as a physical standby database:

```
SQL> ALTER DATABASE MOUNT STANDBY DATABASE;
```

At this point in the switchover process, both databases are configured as standby databases (see Figure 7–3).

### On the target physical standby database

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

After you transition the primary database to the physical standby role and the switchover notification is received by the standby databases in the configuration, you should verify if the switchover notification was processed by the target standby database by querying the SWITCHOVER_STATUS column of the V$DATABASE fixed view on the target standby database.

For example:

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

The SWITCHOVER PENDING value of the SWITCHOVER_STATUS column indicates the standby database is about to switch from the standby role to the primary role. If the SWITCHOVER PENDING value is not displayed, then verify that the Data Guard configuration is functioning correctly (for example, verify that all LOG_ARCHIVE_DEST_*n* parameter values are specified correctly).

> **See Also:** Chapter 14 for information about other valid values for the SWITCHOVER_STATUS column of the V$DATABASE view

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

You can switch a physical standby database from the standby role to the primary role when the standby database instance is either mounted in managed recovery mode or open for read-only access. It must be mounted in one of these modes so that the primary database switchover operation request can be coordinated.

The SQL ALTER DATABASE statement used to perform the switchover automatically creates online redo logs if they do not already exist. This might significantly increase the time required to complete the COMMIT operation. Therefore, Oracle Corporation recommends that you always manually add online redo logs to the target standby database when you create it. Use one of the following methods to manually add the online redo logs if they do not already exist:

- Copy the existing online redo logs from the initial primary database site to the target standby database site and define the LOG_FILE_NAME_CONVERT initialization parameter to correctly associate the standby site path names to the new online redo logs (see Section 3.2.6).

- Drop any existing online redo logs at the target standby site and create new ones using the ALTER DATABASE ADD STANDBY LOGFILE statement.

After you manually add the online redo logs, use the following SQL statement on the physical standby database that you want to transition to the primary role:

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

**Step 6 Shut down and restart the new primary database.**

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

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

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

> **Note:** There is no need to shut down and restart other standby databases (not involved in the switchover) that are online at the time of the switchover operation. These standby databases will continue to function normally after the switchover completes.

**On the new physical standby database**

**Step 7 Start managed recovery operations and log apply services.**

Issue the following statement to begin managed recovery operations on the new physical standby database:

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

**On the new primary database**

**Step 8 Begin sending redo data to the standby databases.**

Issue the following statement on the new primary database:

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

## 7.2.2 Failover Operations Involving a Physical Standby Database

This section describes how to perform failover operations involving a physical standby database.

During failover operations involving a physical standby database:

- In all cases, the original primary database is removed from the Data Guard configuration.
- In most cases, other logical or physical standby databases not directly participating in the failover operation remain in the configuration and do not have to be shut down or restarted.
- In some cases, it might be necessary to re-create all standby databases after configuring the new primary database.

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

#### 7.2.2.1 Failover Steps

The following steps describe how to perform a failover to a physical standby database. Depending on the data protection mode in use, it might be possible to automatically recover all committed transactions.

The steps described in this section transition the selected physical standby database to the primary role so that any other physical or logical standby databases in the configuration will remain in the configuration and will not need to be shut down or restarted.

**Step 1 Identify and resolve any archived redo log gaps.**

To determine if there are gaps at the target standby database in the archived redo logs received from the primary database, query the V$ARCHIVE_GAP view. This view contains the sequence numbers of the archived logs that are known to be missing for each thread. The data returned reflects the highest gap only. (Step 3 will provide instructions on how to resolve any additional gaps.)

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 archive logs 90, 91, and 92 for thread 1. If possible, copy all of the identified missing archived redo logs to the target standby database from the primary database or from another standby database and register them. This must be done for each thread.

For example:

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

### Step 2 Copy any other missing archived redo logs.

To determine if there are any other missing archived redo logs, query the V$ARCHIVED_LOG view on all available databases in the configuration to obtain the highest sequence number for each thread.

For example:

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

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

Copy any archived redo logs from the other available databases that contain sequence numbers higher than the highest sequence number available on the target standby database to the target standby database and register them. This must be done for each for each thread.

For example:

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

---

**Note:** The preceding procedure might result in a *partial* archived redo log being registered. A partial archived redo log contains all of the primary database redo data received by the standby database when the primary database fails, but the archived redo log is not automatically registered in the standby database.

When you register a partial archive log, it prevents the recovery of standby redo logs. Therefore, whether or not you have registered a partial archive log determines which failover command needs to be used. When you manually register a partial archived redo log, the following message is displayed in the alert log:

```
Register archivelog 'filespec1' was created due to
a network disconnect; archivelog contents are
valid but missing subsequent data
```

---

**Step 3 Repeat steps 1 and 2.**

The query executed in step 1 displays information for the highest gap only. After resolving that gap, you must repeat steps 1 and 2 until the query in step 1 returns no rows.

**Step 4 Initiate the failover operation on the target physical standby database.**

If your target standby database was configured with standby redo logs and you have not manually registered any partial archived redo logs, issue the following statement:

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

Otherwise, you must issue the following statement:

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

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

Once the SQL `ALTER DATABASE RECOVER MANAGED STANDBY DATABASE...FINISH` statement completes successfully, transition the physical standby database to the primary database role by issuing the following SQL statement:

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

After issuing this SQL statement, you can no longer use this database as a standby database and subsequent redo logs from the original primary database cannot be applied. The standby redo logs were archived and should be copied to, registered, and recovered on all other standby databases derived from the original primary database. This will happen automatically if the standby destinations are correctly defined on the new primary database.

There is no need to shut down and restart other standby databases in the configuration that were not participants in the failover operation. During a failover, the original primary database is eliminated from participating in the configuration. To reuse the old primary database in the new configuration, you must re-create it as a standby database, using a backup copy of the new primary database.

### On the primary database and all remaining standby databases

#### Step 6 Prepare to receive redo logs from the new primary database.

Once the archived standby redo logs have been received and recovered on all standby destinations, the other standby databases in the configuration are ready to receive redo logs from the new primary database. If the new primary database does not define archived log destinations for the other standby databases, you must define and enable them. In addition, you may need to manually copy and register the resulting failover archived redo logs on each of the remaining standby databases in the configuration.

To manually register any copied archived redo logs that were not automatically applied, issue the following statement on each standby database for each copied log file:

```
SQL> ALTER DATABASE REGISTER LOGFILE 'filespec';
```

### On the new primary database

#### Step 7 Shut down and restart the new primary database.

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

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

> **See Also:** Section 5.8.2 for complete information about how to configure initialization parameters on both the primary and standby databases so that your Data Guard configuration operates properly after a role transition

#### Step 8 Optionally, back up the new primary database.

Optionally, before issuing the STARTUP statement, you might want to back up the new primary database. This task, while not required, is a recommended safety measure, because you cannot recover changes made after the failover without a backup copy.

As a result of the failover operation, the original primary database can no longer participate in the Data Guard configuration, and all other standby databases are now receiving and applying redo data from the new primary database.

# 7.3 Role Transitions Involving Logical Standby Databases

This section describes how to perform switchover and failover operations involving a logical standby database.

## 7.3.1 Switchover Operations Involving a Logical Standby Database

When you perform a switchover operation that changes roles between a primary database and a logical standby database, always initiate the switchover operation on the primary database and complete it on the logical standby database. The following steps describe how to perform the switchover operation.

---

**Note:** Currently, the SWITCHOVER_STATUS column of the V$DATABASE view is supported only for use with physical standby databases and therefore is not queried during switchover operations involving logical standby databases.

---

**On the original primary database**

**Step 1 Switch the primary database to the logical standby database role.**

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

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

This statement waits for all current update operations on the primary database to end and prevents any new users from starting update operations. It also puts a marker in the redo log file to provide a synchronization point for logical standby database operations. Executing this statement also will prevent users from making any changes to the data being maintained in the logical standby database. To ensure faster execution, ensure that 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).

The primary database is transitioned to run in the standby database role.

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

**Step 2 Defer archiving redo logs.**

Defer archiving redo logs for all remote database destinations:

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2=DEFER SCOPE=BOTH;
```

To ensure that this change will persist if the database is later restarted, update the appropriate initialization parameter file or server parameter file. In general, when the database operates in the primary role, you must enable archiving redo logs to remote destinations, and when the database operates in the standby role, you must disable archiving redo logs to remote destinations.

#### On the original logical standby database

#### Step 3 Switch the logical standby database to the primary database role.

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

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

There is no need to shut down and restart any logical standby databases that are in the Data Guard configuration. Other existing logical standby databases will continue to function normally after a switchover operation completes. All existing physical standby databases, however, are rendered unable to participate in the Data Guard configuration after the switchover.

#### Step 4 Enable archiving redo logs.

Identify the initialization parameters that correspond to the new logical standby database and to all other remote logical standby destinations, and enable archiving redo logs for each of these destinations.

> **See Also:** Section 7.1.2.1 which describes how to specify the LOG_ARCHIVE_DEST_*n* and LOG_ARCHIVE_DEST_STATE_*n* initialization parameters for each database to ensure that all standby locations can continue to receive redo data after a role transition

For example, to enable archiving redo logs 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 will persist if the new primary database is later restarted, update the appropriate traditional initialization parameter file or server parameter file. In general, when the database operates in the primary role, you must enable

archiving redo logs to remote destinations, and when the database operates in the standby role, you must disable archiving redo logs to remote destinations.

### On all logical standby databases

#### Step 5 Create a database link to the new primary database.

On each logical standby database (including the former primary database and all other pre-existing logical standby databases), follow these steps to define a database link to the new primary database:

1. On each logical standby database, create a database link that points to the new primary database. (The example in this step uses the database link `location1`.)

   Use the `DBMS_LOGSTDBY.GUARD_BYPASS_ON` procedure to bypass the database guard and allow modifications to the tables in the logical standby database. For example:

   ```
   SQL> EXECUTE DBMS_LOGSTDBY.GUARD_BYPASS_ON;
   SQL> CREATE DATABASE LINK location1
     2> CONNECT TO user-name IDENTIFIED BY password USING 'location1';
   SQL> EXECUTE DBMS_LOGSTDBY.GUARD_BYPASS_OFF;
   ```

   The database user account specified in the `CREATE DATABASE LINK` statement must have the `SELECT_CATALOG_ROLE` role granted to it on the new primary database.

   > **See Also:** *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about the `DBMS_LOGSTDBY` package and the *Oracle9i Database Administrator's Guide* for more information about creating database links

2. Verify the database link.

   On each logical standby database, verify that the database link was configured correctly by executing the following query using the database link:

   ```
   SQL> SELECT * FROM DBA_LOGSTDBY_PARAMETERS@location1;
   ```

   If the query succeeds, then you have verified that the database link created in step 1 can be used to complete the switchover.

#### Step 6 Begin SQL apply operations.

On the new logical standby database (formerly the primary database) and on any other existing logical standby destinations, begin SQL apply operations:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY NEW PRIMARY location1;
```

In the example, `location1` is the database link to the new primary database.

#### On the new primary database

#### Step 7 Ensure all standby databases begin receiving redo logs.

On the new primary database, enable archive logging and switch logs to ensure that all the standby databases begin receiving redo logs by executing the following SQL statements:

```
SQL> ALTER SYSTEM ARCHIVE LOG START;
SQL> ALTER SYSTEM SWITCH LOGFILE;
```

### 7.3.2 Failover Operations Involving a Logical Standby Database

This section describes how to perform failover operations involving a logical standby database.

During failover operations involving a logical standby database:

- In all cases, the original primary database and all physical standby databases are removed from the Data Guard configuration.
- In most cases, other logical standby databases not directly participating in the failover operation remain in the configuration and do not have to be shut down or restarted.
- In some cases, it might be necessary to re-create all standby databases after configuring the new primary database.

Before starting the failover operation, perform as many of the steps documented in Section 7.1.3.1 as possible to prepare the selected standby database for the failover operation. Depending on the protection mode for the configuration and the attributes you chose for log transport services, it might be possible to automatically recover all or some of the primary database modifications.

To start the failover operation, perform the following steps.

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

#### Step 1 Copy and register any missing archived redo logs.

Depending on the nature of the emergency, you might have access to the archived redo logs on the primary database or another standby database. If so, do the following:

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

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

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

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

To resolve the gap in archived redo logs, copy the archived redo logs with sequence numbers 7 and 11 (you copy the files that are one greater than each row shown in the query). Then, register these archived redo logs on the logical standby database. For example:

```
SQL> ALTER DATABASE REGISTER LOGICAL LOGFILE
  2> '/disk1/oracle/dbs/log-1292880008_7.arc';
Database altered.

SQL> ALTER DATABASE REGISTER LOGICAL LOGFILE
  2> '/disk1/oracle/dbs/log-1292880008_11.arc';
```

```
Database altered.
```

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

### Step 2 Copy and register the online redo logs from the primary database.

Depending on the nature of the emergency, online redo logs might be available on the primary database. If so, you should copy and register the missing online redo logs from the primary database and then apply them to the logical standby database.

If you register any online redo logs that have already been archived and registered on the logical standby database, the ORA-01289 message is returned. You can safely ignore this error message. For example:

```
SQL> ALTER DATABASE REGISTER LOGICAL LOGFILE
  2> '/disk1/oracle/dbs/online_log1.log';
ALTER DATABASE REGISTER LOGICAL LOGFILE '/disk1/oracle/dbs/online_log1.log'
*
ERROR at line 1:
ORA-01289: cannot add duplicate logfile

SQL> ALTER DATABASE REGISTER LOGICAL LOGFILE
  2> '/disk1/oracle/dbs/online_log2.log';
Database altered.
```

### Step 3 Register partially filled archived redo logs, if any exist.

Depending on the nature of the emergency, you might not have access to any files on the primary database. To look for a partially filled archived redo log, query the DBA_LOGSTDBY_LOG view on the logical standby database in the same directory where the other archived redo logs are located. If a partially filled archived redo log exists, its sequence number will be one greater than the last registered archived redo log. For example:

```
SQL> COLUMN FILE_NAME FORMAT a55
SQL> SELECT THREAD#, SEQUENCE#, FILE_NAME FROM DBA_LOGSTDBY_LOG L
  2> ORDER BY THREAD#,SEQUENCE#;

   THREAD#  SEQUENCE# FILE_NAME
---------- ---------- -------------------------------------------------------
         1          3 /disk1/oracle/dbs/db1loga-1292880008_3.arc
         1          4 /disk1/oracle/dbs/archlogb-1292880008_4.arc
```

```
         1          5 /disk1/oracle/dbs/archlogb-1292880008_5.arc
         1          6 /disk1/oracle/dbs/archlogb-1292880008_6.arc
         1          7 /disk1/oracle/dbs/archlogb-1292880008_7.arc
         1          8 /disk1/oracle/dbs/archlogb-1292880008_8.arc
         1          9 /disk1/oracle/dbs/archlogb-1292880008_9.arc
         1         10 /disk1/oracle/dbs/archlogb-1292880008_10.arc

8 rows selected.
```

If a partially filled archived log exists, register it. For example:

```
SQL> ALTER DATABASE REGISTER LOGICAL LOGFILE
  2> '/disk1/oracle/dbs/log-1292880008_11.arc';

Database altered.
```

### Step 4 Turn off the apply delay interval.

To turn off the apply delay interval, stop log apply services, and execute the DBMS_LOGSTDBY.APPLY_UNSET procedure. Then, restart log apply services.

Although the apply delay interval might have been set originally on the primary database, you must issue the following statements on the logical standby database to disable the apply delay interval because the primary database is no longer available:

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

SQL> EXECUTE DBMS_LOGSTDBY.APPLY_UNSET('APPLY_DELAY');
PL/SQL procedure successfully completed.

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

### Step 5 Ensure that all redo logs were applied.

On the logical standby database that you are transitioning to the primary role, verify that the remaining archived redo logs were applied by querying the DBA_LOGSTDBY_PROGRESS view. For example:

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

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

When the APPLIED_SCN and NEWEST_SCN values are equal, all attainable data is applied and the logical standby database now contains as much data as possible from the primary database.

> **See Also:** Chapter 9 and Chapter 10 for information about the DBA_LOGSTDBY_PROGRESS view

#### Step 6 Activate the new primary database.

Issue the following statements on the logical standby database (that you are transitioning to the new primary role) to stop SQL apply operations and activate the database in the primary database role:

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

### On all other logical standby databases

#### Step 1 Recover the other standby databases.

Depending on how much redo data you were able to apply to the new primary database, you might or might not be able to add existing logical standby databases back into the Data Guard configuration to serve as standby databases for the new primary database.

#### Step 2 Create a database link to the new primary database from the other standby databases.

Follow these steps to define a database link to the new primary database that will be used during future switchover operations:

1. Create a database link on each logical standby database.

   Use the DBMS_LOGSTDBY.GUARD_BYPASS_ON procedure to bypass the database guard and allow modifications to the tables in the logical standby database. For example:

   ```
   SQL> EXECUTE DBMS_LOGSTDBY.GUARD_BYPASS_ON;
   SQL> CREATE DATABASE LINK location1
     2> CONNECT TO <user-name> IDENTIFIED BY <password> USING 'location1';
   SQL> EXECUTE DBMS_LOGSTDBY.GUARD_BYPASS_OFF;
   ```

   The database user account specified in the CREATE DATABASE LINK statement must have the SELECT_CATALOG_ROLE role granted to it on the primary database.

> **See Also:** *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about the DBMS_LOGSTDBY package and *Oracle9i Database Administrator's Guide* for more information about creating database links.

2. Verify the database link.

   On the logical standby database, verify that the database link was configured correctly by executing the following query using the database link:

   ```
   SQL> SELECT * FROM DBA_LOGSTDBY_PARAMETERS@location1;
   ```

   If the query succeeds, then you have verified that the database link created in step 1 can be used to perform a switchover.

### On the new primary database

Enable archiving redo logs to all remote logical standby destinations. For example:

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2=ENABLE SCOPE=BOTH;
```

To ensure that this change will persist if the new primary database is later restarted, update the appropriate initialization parameter file or server parameter file. In general, when the database operates in the primary role, you must enable archiving redo logs to remote destinations, and when the database operates in the standby role, you must disable archiving redo logs to remote destinations.

### On all logical standby databases

Begin log apply services by issuing this SQL statement on all logical standby databases:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY NEW PRIMARY location1;
```

When this statement completes, all remaining archived redo logs will have been applied. Depending on the work to be done, this operation can take some time to complete.

If the ORA-16109 error is returned, you must re-create the logical standby database from a backup copy of the new primary database, and then add it to the Data Guard configuration.

The following example shows a failed attempt to start log apply services on a logical standby database in the new configuration where `location1` points to the new primary database:

```
SQL> ALTER DATABASE START LOGICAL STANDBY APPLY NEW PRIMARY location1;
ALTER DATABASE START LOGICAL STANDBY APPLY NEW PRIMARY location1
                                                       *
ERROR at line 1:
ORA-16109: failed to apply log data from previous primary
```

# 8

# Managing a Physical Standby Database

This chapter describes how to manage a physical standby database. Data Guard provides the means to easily manage, manipulate, and change a physical standby database in many ways.

This chapter contains the following topics:



## 8.1 Starting Up and Shutting Down a Physical Standby Database

This section describes the procedures for starting up and shutting down a physical standby database.

### 8.1.1 Starting Up a Physical Standby Database

To start up a physical standby database, use SQL*Plus to connect to the database with administrator privileges, and then use the SQL*Plus `STARTUP` command with the `NOMOUNT` option. (You must use the `NOMOUNT` option with a standby database.)

If both the primary and standby databases are offline, then always (if possible) start the standby database before starting the primary database.

After the database is started, mount the database as a standby database. Once it is mounted, the database can receive archived redo data from the primary database.

You then have the option of either starting a managed recovery operation or opening the database for read-only access. Typically, you start a managed recovery operation. The following example shows how to start a standby database:

1. Start the database:

```
SQL> STARTUP NOMOUNT;
```

2. Mount the standby database:

```
SQL> ALTER DATABASE MOUNT STANDBY DATABASE;
```

3. Start the managed recovery operation:

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

Once the database is performing managed recovery, log apply services apply the archived redo logs to the standby database.

> **See Also:** Section 6.2.2 for information about managed recovery and Section 8.2 for information on opening a standby database for read-only access

## 8.1.2 Shutting Down a Physical Standby Database

To shut down a physical standby database, use the SQL*Plus SHUTDOWN command. If the database is performing managed recovery, you must cancel managed recovery operations before issuing the SHUTDOWN command. 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 archive log destination on the primary database and perform a log switch operation (to make the defer operation take effect) before shutting down the standby database. Otherwise, log transport services will not be able to transmit redo data to this standby site.

The following steps show you how to shut down a standby database:

1. Find out if the standby database is performing managed recovery. If the MRP0 or MRP process exists, then the standby database is performing managed recovery.

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

2. Cancel managed recovery operations.

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

3. Shut down the standby database.

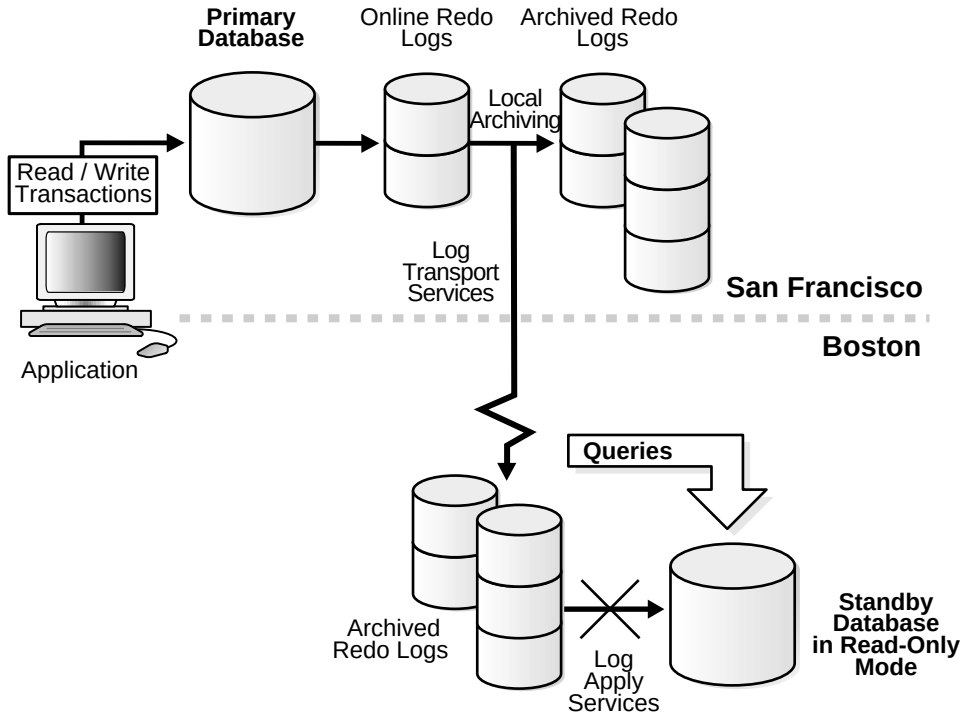
```
SQL> SHUTDOWN IMMEDIATE;
```

## 8.2 Using a Standby Database That Is Open for Read-Only Access

When a standby database is open for read-only access, users can query the standby database without the potential for online data modifications. This reduces the load on the primary database by using the standby database for reporting purposes. You can periodically open the standby database for read-only access and perform ad hoc queries to ensure that log apply services are updating the standby database correctly.

Figure 8–1 shows a standby database open for read-only access.

***Figure 8–1 Standby Database Open for Read-Only Access***



This section contains the following topics:



## 8.2.1 Assessing Whether to Open a Standby Database for Read-Only Access

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

- Having a physical standby database open for read-only access makes it unavailable for managed recovery operations. The archived redo data is received by the standby database, but the redo logs are not applied. Therefore, a standby database that is open for read-only access is not transactionally current with the primary database. At some point, you need to resume managed recovery on the standby database, and apply the archived redo logs to resynchronize the standby database with the primary database. Having a standby database open for read-only access might prolong a failover or switchover operation if one is required for disaster recovery.
- If you need a standby database for protection against disaster and for reporting, then you can maintain multiple standby databases: some open for read-only access and some performing managed recovery (which will automatically apply the archived redo logs to the standby database). However, you will need to perform managed recovery on all the standby databases periodically to ensure that the latest changes from the primary database are applied to the standby database. A physical standby database performing managed recovery gives you immediate protection against disaster.

---

**Note:** Consider using a logical standby database if your business requires that the standby database be used for queries and reporting purposes concurrent with fulfilling disaster recovery requirements.

---

## 8.2.2 Opening a Standby Database for Read-Only Access

You can alternate between having a standby database open for read-only access and having a standby database perform managed recovery using the following procedures.

**To open a standby database for read-only access when it is currently shut down:**

1. Start the Oracle instance for the standby database without mounting it:

```
SQL> STARTUP NOMOUNT;
```

2. Mount the standby database:

```
SQL> ALTER DATABASE MOUNT STANDBY DATABASE;
```

3. Open the database for read-only access:

```
SQL> ALTER DATABASE OPEN READ ONLY;
```

**To open a standby database for read-only access when it is currently performing managed recovery:**

1. Cancel log apply services:

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

2. Open the database for read-only access:

```
SQL> ALTER DATABASE OPEN READ ONLY;
```

**To change the standby database from being open for read-only access to performing managed recovery:**

1. Terminate all active user sessions on the standby database.
2. Restart log apply services:

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

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

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

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

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

To perform queries that sort a large amount of data on a standby database that is open for read-only access, the Oracle database server must be able to perform

on-disk sorting operations. You cannot allocate space for sorting operations in tablespaces that cause Oracle software to write to the data dictionary.

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

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

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

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

1. Enter the following SQL statement:

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

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

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

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

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

On the physical standby database, perform the following steps:

1. Start managed recovery, if necessary, and apply the archived redo logs by entering the following SQL statement:

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

2. Cancel managed recovery and open the physical standby database for read-only access using the following SQL statements:

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

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

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

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

**See Also:** *Oracle9i SQL Reference* for information about the CREATE TEMPORARY TABLESPACE syntax

#### 8.2.3.2 Sorting Operations Without Temporary Tablespaces

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

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

select * from v$parameter

              *

ERROR at line 1:

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

Note that you can, however, sort small amounts of data if the SORT_AREA_SIZE parameter is set to a sufficient value in your server parameter file. (The SORT_AREA_SIZE parameter is a static parameter.)

## 8.3 Creating Primary Database Back Up Files Using a Physical Standby Database

You can use the physical standby database to off-load the database backup operation from the primary database because a physical standby database is a copy of the primary database. Using RMAN at the standby site, you can back up the datafiles and the archived redo logs while the standby database is performing managed recovery. Later, you can restore these backups to the primary database using RMAN.

---

**Note:** You cannot use a logical standby database to back up the primary database.

---

**See Also:** *Oracle9i Recovery Manager User's Guide* for more details about RMAN backup and recovery of a primary database using a standby database

## 8.4 Managing Primary Database Events That Affect the Standby Database

To prevent possible problems, you should be aware of events in the primary database that affect a standby database and learn how to respond to them. This section describes these events and the recommended responses to these events.

In some cases, the events or changes that occur on a primary database are automatically propagated through archived redo logs to the standby database and thus require no extra action on the standby database. In other cases, you might need to perform maintenance tasks on the standby database.

Table 8–1 indicates whether or not a change made on the primary database requires additional intervention by the database administrator (DBA) to be propagated to the standby database. It also briefly describes how to respond to these events. Detailed descriptions of the responses are described in the section references provided.

**Caution:** If you clear logs at the primary database by issuing the ALTER DATABASE CLEAR UNARCHIVED LOGFILE statement, or open the primary database using the RESETLOGS option, you invalidate the standby database. Because both of these operations reset the primary log sequence number to 1, you must re-create the standby database to be able to apply archived redo logs generated by the primary database.

The following events are automatically administered by log transport services and log apply services, and therefore require no intervention by the database administrator:

- A SQL ALTER DATABASE statement is issued with the ENABLE THREAD or DISABLE THREAD clause.
- The status of a tablespace changes (changes to read/write or read-only, placed online or taken offline).
- A datafile is added or tablespace is created when the STANDBY_FILE_MANAGEMENT initialization parameter is set to AUTO.

***Table 8–1 Actions Required on a Standby Database After Changes to a Primary Database***

| Reference | Change Made on Primary Database | Action Required on Standby Database |
|---|---|---|
| Section 8.4.1 | Add a datafile or create a tablespace | If you did not set the STANDBY_FILE_MANAGEMENT initialization parameter to AUTO, you must copy the new datafile to the standby database. |
| Section 8.4.2 | Drop or delete a tablespace or datafile | Delete the corresponding datafile after the archived redo log was applied. |
| Section 8.4.3 | Rename a datafile | Rename the datafile on the standby database. |
| Section 8.4.4 | Add or drop online redo logs | Synchronize changes on the standby database. |
| Section 8.4.5 | Alter the primary database control file (using the SQL ALTER DATABASE CREATE CONTROLFILE statement) | Re-create the standby control file or re-create the standby database, depending on the alteration made. |
| Section 8.4.6 | Perform a DML or DDL operation using the NOLOGGING or UNRECOVERABLE clause | Send the datafile containing the unlogged changes to the standby database. |
| Chapter 11 | Change initialization parameter | Dynamically change the standby parameter or shut down the standby database and update the initialization parameter file. |

## 8.4.1 Adding a Datafile or Creating a Tablespace

The initialization parameter, STANDBY_FILE_MANAGEMENT, allows you to control whether or not adding a datafile to the primary database is automatically propagated to the standby database, as follows:

- If you set the STANDBY_FILE_MANAGEMENT initialization parameter in the standby database server parameter file to AUTO, any new datafiles created on the primary database are automatically created on the standby database as well.
- If you do not specify the STANDBY_FILE_MANAGEMENT initialization parameter or if you set it to MANUAL, then you must manually copy the new datafile to the standby database when you add a datafile to the primary database.

Note that if you copy an existing datafile from another database to the primary database, then you must also copy the new datafile to the standby database and re-create the standby control file, regardless of the setting of STANDBY_FILE_MANAGEMENT initialization parameter.

The following sections provide examples of adding a datafile to the primary and standby databases when the STANDBY_FILE_MANAGEMENT initialization parameter is set to AUTO and MANUAL, respectively.

### 8.4.1.1 Adding a Tablespace and a Datafile When STANDBY_FILE_MANAGEMENT Is Set to AUTO

The following example shows the steps required to add a new datafile to the primary and standby databases when the STANDBY_FILE_MANAGEMENT initialization parameter is set to AUTO.

1. Add a new tablespace to the primary database:

```
SQL> CREATE TABLESPACE new_ts DATAFILE 't_db2.dbf'
  2> SIZE 1m AUTOEXTEND ON MAXSIZE UNLIMITED;
```

2. Archive the current redo log so it will get copied to the standby database:

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

3. Verify that the new datafile was added to the primary database:

```
SQL> SELECT NAME FROM V$DATAFILE;
NAME
--------------------------------------------------------------------------
/disk1/oracle/dbs/t_db1.dbf
/disk1/oracle/dbs/t_db2.dbf
```

4. Verify that the new datafile was added to the standby database:

```
SQL> SELECT NAME FROM V$DATAFILE;
NAME
----------------------------------------------------------------------
/disk1/oracle/dbs/s2t_db1.dbf
/disk1/oracle/dbs/s2t_db2.dbf
```

#### 8.4.1.2 Adding a Tablespace and a Datafile When STANDBY_FILE_MANAGEMENT Is Set to MANUAL

The following example shows the steps required to add a new datafile to the primary and standby database when the STANDBY_FILE_MANAGEMENT initialization parameter is set to MANUAL. You must set the STANDBY_FILE_MANAGEMENT initialization parameter to MANUAL when the standby datafiles reside on raw devices.

1. Add a new tablespace to the primary database:

```
SQL> CREATE TABLESPACE new_ts DATAFILE 't_db2.dbf'
  2> SIZE 1m AUTOEXTEND ON MAXSIZE UNLIMITED;
```

2. Verify that the new datafile was added to the primary database:

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

----------------------------------------------------------------------
/disk1/oracle/dbs/t_db1.dbf
/disk1/oracle/dbs/t_db2.dbf
```

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

   a. Place the new tablespace offline:

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

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

```
% cp t_db2.dbf s2t_db2.dbf
```

   c. Place the new tablespace back online:

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

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

```
%rcp s2t_db2.dbf standby_location
```

**4.** Archive the current redo log on the primary database so it will get copied to the standby database:

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

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

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

**6.** Verify that the datafile was added to the standby database after the redo log was applied to the standby database.

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

----------------------------------------------------------------------
/disk1/oracle/dbs/s2t_db1.dbf
/disk1/oracle/dbs/s2t_db2.dbf
```

## 8.4.2 Dropping a Tablespace in the Primary Database

When you delete one or more datafiles or drop one or more tablespaces in the primary database, you also need to delete the corresponding datafiles in the standby database, as follows:

**1.** Drop the tablespace at the primary site:

```
SQL> DROP TABLESPACE tbs_4;
SQL> ALTER SYSTEM SWITCH LOGFILE;
% rm tbs_4.dbf
```

**2.** Make sure that managed recovery is on (so that the change is applied to the standby database). If the following query returns the MRP or MRP0 process, managed recovery is on.

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

3. Delete the corresponding datafile on the standby site after the archived redo log was applied to the standby database. For example:

```
% rm tbs_4.dbf
```

4. On the primary database, after ensuring that the standby database has applied the redo information for the dropped tablespace, you can remove the datafile for the tablespace. For example:

```
% rm tbs_4.dbf
```

### 8.4.3 Renaming a Datafile in the Primary Database

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

The following steps describe how to rename a datafile in the primary database and manually propagate the changes to the standby database. If you do not want the standby database to have the same physical structure as the primary database, then these steps are not required.

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

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

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

```
% mv tbs_4.dbf tbs_x.dbf
```

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

```
SQL> ALTER TABLESPACE tbs_4 RENAME DATAFILE 'tbs_4.dbf'
  2>                                     TO 'tbs_x.dbf';
SQL> ALTER TABLESPACE tbs_4 ONLINE;
```

4. Connect to the standby database and make sure that all the logs are applied; then stop managed recovery operations:

```
SQL> SELECT NAME, SEQUENCE#, ARCHIVED, APPLIED
  2> FROM V$ARCHIVED_LOG;
```

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

5. Shut down the standby database:

```
SQL> SHUTDOWN;
```

6. Rename the datafile at the standby site using an operating system command, such as the UNIX mv command:

```
% mv tbs_4.dbf tbs_x.dbf
```

7. Start and mount the standby database with the new control file:

```
SQL> STARTUP NOMOUNT;
SQL> ALTER DATABASE MOUNT STANDBY DATABASE;
```

8. Rename the datafile in the standby controlfile. Note that the STANDBY_FILE_MANAGEMENT initialization parameter must be set to MANUAL.

```
SQL> ALTER DATABASE RENAME FILE 'tbs_4.dbf'
  2>                         TO 'tbs_x.dbf';
```

9. On the standby database, restart managed recovery operations:

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

If you do not rename the corresponding datafile at the standby site, and then try to refresh the standby database control file, the standby database will attempt to use the renamed datafile, but it will not find it. Consequently, you will see error messages similar to the following in the alert log:

```
ORA-00283: recovery session canceled due to errors
ORA-01157: cannot identify/lock data file 4 - see DBWR trace file
ORA-01110: data file 4: '/disk1/oracle/dbs/tbs_x.dbf'
```

### 8.4.4 Adding or Dropping Online Redo Logs

Changing the size and number of the online redo logs is sometimes done to tune the database. You can add redo log file groups or members to the primary database without affecting the standby database. Similarly, you can drop log file groups or members from the primary database without affecting your standby database. However, these changes do affect the performance of the standby database after switchover.

For example, if the primary database has 10 redo logs and the standby database has 2, and then you switch over to the standby database so that it functions as the new primary database, the new primary database is forced to archive more frequently than the original primary database.

Consequently, when you add or drop an online redo log at the primary site, it is important that you synchronize the changes in the standby database by following these steps:

1. If managed recovery is on, you must cancel it before you can change the logs.
2. If the STANDBY_FILE_MANAGEMENT initialization parameter is set to AUTO, change the value to MANUAL.
3. Add or drop an online redo log:
   - To add an online redo log, use a SQL statement such as this:

     ```
     SQL> ALTER DATABASE ADD STANDBY LOGFILE 'prmy3.log' SIZE 100K;
     ```

   - To drop an online redo log, use a SQL statement such as this:

     ```
     SQL> ALTER DATABASE DROP STANDBY LOGFILE 'prmy3.log';
     ```

4. Repeat the statement you used in step 3 on each standby database.
5. Restore the STANDBY_FILE_MANAGEMENT initialization parameter and the managed recovery options to their original states.

## 8.4.5 Altering the Primary Database Control File

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

If you invalidated the control file for the standby database, re-create the file using the procedure provided in Section 3.2.3.

If you invalidated the standby database, you must re-create the standby database using the procedures in Chapter 3.

## 8.4.6 NOLOGGING or Unrecoverable Operations

When you perform a DML or DDL operation using the NOLOGGING or UNRECOVERABLE clause, the standby database is invalidated and might require

substantial DBA administrative activities to repair. You can specify the SQL ALTER DATABASE or SQL ALTER TABLESPACE statement with the FORCELOGGING clause to override the NOLOGGING setting. However, this statement will not repair an invalidated database.

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

> **See Also:** Section 10.5 for information about recovering after the NOLOGGING clause is used

## 8.5 Monitoring the Primary and Standby Databases

This section gives you a general overview on where to find information for monitoring the primary and standby databases in a Data Guard environment.

This section contains the following topics:

- Alert Log
- Dynamic Performance Views (Fixed Views)
- Monitoring Recovery Progress

Table 8–2 summarizes common events that occur on the primary database and pointers to the files and views where you can monitor these events on the primary and standby sites.

***Table 8–2 Location Where Common Actions on the Primary Database Can Be Monitored***

| Primary Database Event | Primary Site Information | Standby Site Information |
|---|---|---|
| A SQL ALTER DATABASE statement is issued with the ENABLE THREAD or DISABLE THREAD clause specified | ■ Alert log<br>■ V$THREAD view | Alert log |
| Redo log changed | ■ Alert log<br>■ V$LOG view<br>■ STATUS column of V$LOGFILE view | Alert log |
| CREATE CONTROLFILE statement issued | Alert log | Alert log[1] |

***Table 8–2 Location Where Common Actions on the Primary Database Can Be Monitored***

| Primary Database Event | Primary Site Information | Standby Site Information |
|---|---|---|
| Managed recovery performed | Alert log | Alert log |
| Tablespace status changes made (made read / write or read-only, placed online or offline) | ■ DBA_TABLESPACES view<br>■ Alert log | V$RECOVER_FILE view |
| Datafile added or tablespace created | ■ DBA_DATA_FILES view<br>■ Alert log | V$DATAFILE view<br>Alert log |
| Tablespace dropped | ■ DBA_DATA_FILES view<br>■ Alert log | V$DATAFILE view<br>Alert log |
| Tablespace or datafile taken offline, or datafile is deleted offline | ■ V$RECOVER_FILE view<br>■ Alert log | V$RECOVER_FILE view |
| Rename datafile | ■ V$DATAFILE<br>■ Alert log | V$DATAFILE<br>Alert log |
| Unlogged or unrecoverable operations | ■ V$DATAFILE view<br>■ V$DATABASE view | Alert log |
| Recovery progress | ■ V$ARCHIVE_DEST_STATUS view<br>■ Alert log | V$ARCHIVED_LOG view<br>V$LOG_HISTORY view<br>V$MANAGED_STANDBY view<br>Alert log |
| Autoextend a datafile | Alert log | Alert log |
| Issue OPEN RESETLOGS or CLEAR UNARCHIVED LOGFILES statements | Alert log | Alert log |
| Change initialization parameter | Alert log | Alert log |

[1] When you issue a CREATE CONTROLFILE statement on the primary database, the standby database functions normally until it encounters redo data that depends on initialization parameters.

## 8.5.1 Alert Log

The database alert log is a chronological record of messages and errors. Besides providing information about the Oracle database, it also includes information about operations specific to Data Guard, including the following:

- Messages related to administrative operations such as the following SQL statements: `ALTER DATABASE RECOVER MANAGED STANDBY`, `STARTUP`, `SHUTDOWN`, `ARCHIVE LOG`, and `RECOVER`
- Errors related to administrative operations that are reported by background processes, such as ARC0, MRP0, RFS, LGWR
- The completion time stamp for administrative operations

The alert log also provides pointers to the trace or dump files generated by a specific process.

## 8.5.2 Dynamic Performance Views (Fixed Views)

The Oracle database server contains a set of underlying views that are maintained by the server. These views are often called **dynamic performance views** because they are continuously updated while a database is open and in use, and their contents relate primarily to performance. These views are also called **fixed views** because they cannot be altered or removed by the database administrator.

These view names are prefixed with either V$ or GV$, for example, `V$ARCHIVE_DEST` or `GV$ARCHIVE_DEST`.

Standard dynamic performance views (V$ fixed views) store information on the local instance. In contrast, global dynamic performance views (GV$ fixed views), store information on all open instances. Each V$ fixed view has a corresponding GV$ fixed view.

> **See Also:** Chapter 14, "Views" and the *Oracle9i Database Reference* for additional information on view columns

## 8.5.3 Monitoring Recovery Progress

This section shows some samples of the types of views discussed in Section 8.5.2 for monitoring recovery progress in a Data Guard environment. It contains the following examples:

- Monitoring the Process Activities
- Determining the Progress of Managed Recovery Operations
- Determining the Location and Creator of Archived Redo Logs
- Viewing the Archive Log History
- Determining Which Logs Were Applied to the Standby Database

- Determining Which Logs Were Not Received by the Standby Site

#### 8.5.3.1 Monitoring the Process Activities

You can obtain information about managed recovery operations on a standby database by monitoring the activities performed by the following processes:

- ARC0
- MRP / MRP0
- RFS

The V$MANAGED_STANDBY view on the standby database site shows you the activities performed by both log transport and log apply processes in a Data Guard environment. The CLIENT_P column in the output of the following query identifies the corresponding primary database process.

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

PROCESS CLIENT_P  SEQUENCE# STATUS
------- -------- ---------- ------------
ARCH    ARCH              0 CONNECTED
ARCH    ARCH              0 CONNECTED
MRP0    N/A             204 WAIT_FOR_LOG
RFS     LGWR            204 WRITING
RFS     N/A               0 RECEIVING
```

#### 8.5.3.2 Determining the Progress of Managed Recovery Operations

The V$ARCHIVE_DEST_STATUS view on either a primary or standby database site provides you information such as the redo logs that are archived, the archived redo logs that are applied, and the log sequence numbers of each. The following query output shows the standby database is two archived logs behind in applying the redo logs received from the primary database.

```
SQL> SELECT ARCHIVED_THREAD#, ARCHIVED_SEQ#, APPLIED_THREAD#, APPLIED_SEQ#
  2> FROM V$ARCHIVE_DEST_STATUS;

ARCHIVED_THREAD# ARCHIVED_SEQ# APPLIED_THREAD# APPLIED_SEQ#
---------------- ------------- --------------- ------------
1                947           1               945
```

#### 8.5.3.3 Determining the Location and Creator of Archived Redo Logs

You can also query the V$ARCHIVED_LOG view on the standby database to find additional information about archived redo logs. Some information you can get

includes the location of the archived redo log, which process created the archived redo log, redo log sequence number of each archived redo log, when the log was archived, and whether or not the archived redo log was applied. For example:

```
SQL> SELECT NAME, CREATOR, SEQUENCE#, APPLIED, COMPLETION_TIME
  2> FROM V$ARCHIVED_LOG;

NAME                                             CREATOR SEQUENCE# APP COMPLETIO
-----------------------------------------------  ------- --------- --- ---------
H:\ORACLE\ORADATA\PAYROLL\STANDBY\ARC00198.001  FGRD          198 YES 30-MAY-02
H:\ORACLE\ORADATA\PAYROLL\STANDBY\ARC00199.001  FGRD          199 YES 30-MAY-02
H:\ORACLE\ORADATA\PAYROLL\STANDBY\ARC00200.001  FGRD          200 YES 30-MAY-02
H:\ORACLE\ORADATA\PAYROLL\STANDBY\ARC00201.001  LGWR          201 YES 30-MAY-02
H:\ORACLE\ORADATA\PAYROLL\STANDBY\ARC00202.001  FGRD          202 YES 30-MAY-02
H:\ORACLE\ORADATA\PAYROLL\STANDBY\ARC00203.001  LGWR          203 YES 30-MAY-02

6 rows selected.
```

### 8.5.3.4 Viewing the Archive Log History

The V$LOG_HISTORY on the physical standby site shows you a complete history of the archived log, including information such as the time of the first entry, the lowest SCN in the log, the highest SCN in the log, and the sequence number of the archived log.

```
SQL> SELECT FIRST_TIME, FIRST_CHANGE#, NEXT_CHANGE#, SEQUENCE# FROM V$LOG_
HISTORY;

FIRST_TIM FIRST_CHANGE# NEXT_CHANGE#  SEQUENCE#
--------- ------------- ------------ ----------
13-MAY-02        190578       214480          1
13-MAY-02        214480       234595          2
13-MAY-02        234595       254713          3
.
.
.
30-MAY-02       3418615      3418874        201
30-MAY-02       3418874      3419280        202
30-MAY-02       3419280      3421165        203

203 rows selected.
```

### 8.5.3.5 Determining Which Logs Were Applied to the Standby Database

Query the V$LOG_HISTORY view on the standby database, which records the latest log sequence number that was applied. For example, issue the following query:

```
SQL> SELECT THREAD#, MAX(SEQUENCE#) AS "LAST_APPLIED_LOG"
  2> FROM V$LOG_HISTORY
  3> GROUP BY THREAD#;

THREAD# LAST_APPLIED_LOG
------- ----------------
      1              967
```

In this example, the archived redo log with log sequence number 967 is the most recently applied log.

You can also use the APPLIED column in the V$ARCHIVED_LOG fixed view on the standby database to find out which log is applied on the standby database. The column displays YES for the log that was applied. For example:

```
SQL> SELECT THREAD#, SEQUENCE#, APPLIED FROM V$ARCHIVED_LOG;

   THREAD#  SEQUENCE# APP
---------- ---------- ---
         1          2 YES
         1          3 YES
         1          4 YES
         1          5 YES
         1          6 YES
         1          7 YES
         1          8 YES
         1          9 YES
         1         10 YES
         1         11 NO

10 rows selected.
```

### 8.5.3.6 Determining Which Logs Were Not Received by the Standby Site

Each archive destination has a destination ID assigned to it. You can query the DEST_ID column in the V$ARCHIVE_DEST fixed view to find out your destination ID. You can then use this destination ID in a query on the primary database to discover logs that were not sent to a particular standby site.

For example, assume the current local archive destination ID on your primary database is 1, and the destination ID of one of your remote standby databases is 2.

To find out which logs were not received by this standby destination, issue the following query on the primary database:

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

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

The preceding example shows the logs that were not received by standby destination 2.

# 9

# Managing a Logical Standby Database

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

- Configuring and Managing Logical Standby Databases
- Tuning Logical Standby Databases

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

> **See Also:** *Oracle9i Data Guard Broker* to use the Data Guard broker to automate the management tasks described in this chapter

## 9.1 Configuring and Managing Logical Standby Databases

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

- Managing SQL Apply Operations
- Controlling User Access to Tables in a Logical Standby Database
- Modifying a Logical Standby Database
- Handling Triggers and Constraints on a Logical Standby Database
- Skipping SQL Apply Operations on a Logical Standby Database
- Adding or Re-Creating Tables on a Logical Standby Database
- Viewing and Controlling Logical Standby Events

- Viewing SQL Apply Operations Activity
- Delaying the Application of Archived Redo Logs
- Determining How Much Redo Log Data Was Applied
- Recovering from Errors
- Refreshing Materialized Views

---

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

---

### 9.1.1 Managing SQL Apply Operations

The DBMS_LOGSTDBY PL/SQL package includes procedures to help you manage SQL apply operations on logical standby databases. Using it you can do the following:

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

> **See Also:** *Oracle9i Supplied PL/SQL Packages and Types Reference* for complete information about the DBMS_LOGSTDBY package and Chapter 9 for a summary of the procedures provided by the DBMS_LOGSTDBY PL/SQL package

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

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

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

**See Also:** *Oracle9i Supplied PL/SQL Packages and Types Reference* for complete information about the DBMS_LOGSTDBY package

## 9.1.2 Controlling User Access to Tables in a Logical Standby Database

The SQL ALTER DATABASE GUARD statement controls user access to tables in logical standby databases. Until you start log apply services on the logical standby

database, users can modify the logical standby database. However, once you start log apply services, the database guard is set to ALL by default.

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 operations.

- NONE

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

For example, use the following statement to enable the database guard and prevent user access to tables in the logical standby database:

```
SQL> ALTER DATABASE GUARD ALL;
```

## 9.1.3 Modifying a Logical Standby Database

You can temporarily override the database guard to allow changes to the logical standby database by executing the DBMS_LOGSTDBY.GUARD_BYPASS_ON procedure. The following sections describe two examples that show when it might be useful to bypass the database guard temporarily to make changes to the logical standby database:

- Performing DDL on a Logical Standby Database
- Modifying Tables That Are Not Maintained by SQL Apply

The discussions in these sections assume that the database guard is set to ALL or STANDBY.

### 9.1.3.1 Performing DDL on a Logical Standby Database

This section describes how to add an index to a table maintained through SQL apply operations.

By default, only accounts with SYS privileges can modify the database while the database guard is set to ALL or STANDBY. If you are logged in as SYSTEM or another

privileged account, you will not be able to issue DDL statements on the logical standby database without first bypassing the database guard for the session.

The following example shows how to stop log apply services, bypass the database guard, execute SQL statements on the logical standby database, and then reenable the guard:

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

SQL> EXECUTE DBMS_LOGSTDBY.GUARD_BYPASS_ON;
PL/SQL procedure successfully completed.

SQL> ALTER TABLE SCOTT.EMP ADD CONSTRAINT EMPID UNIQUE (EMPNO);
Table altered.

SQL> EXECUTE DBMS_LOGSTDBY.GUARD_BYPASS_OFF;
PL/SQL procedure successfully completed.

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

This sample procedure could be used to execute other DDL statements. Oracle Corporation recommends that you do not perform DML operations while the database guard bypass is enabled. This will introduce deviations between the primary and standby databases that will make it impossible for the logical standby database to be maintained. It is unlikely that you will be able to modify rows in a table in such a way that the logical standby database can incrementally maintain the rows.

#### 9.1.3.2 Modifying Tables That Are Not Maintained by SQL Apply

Sometimes, a reporting application must collect summary results and store them temporarily or track the number of times a report was run. Although the main purpose of an application is to perform reporting activities, the application might need to issue DML (insert, update, and delete) operations on a logical standby database. It might even need to create or drop tables.

You can set up the database guard to allow reporting operations to modify data as long as the data is not being maintained through SQL apply operations. 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 operations.

- Set the database guard to protect only standby tables. This setting describes the list of tables that the logical standby database is maintaining. The list cannot include the tables to which your application will be writing.

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 operations, skips the tables, and then restarts SQL apply operations so that changes can be applied to the logical standby database. The reporting application will be able to write to MYTABLES% in MYSCHEMA. They will no longer be maintained through SQL apply operations.

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

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

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

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

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

```
SQL> SELECT NAME FROM DBA_LOGSTDBY_PARAMETERS WHERE NAME = 'EVALUATE_SKIP';
no rows selected
```

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

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

### 9.1.4 Handling Triggers and Constraints on a Logical Standby Database

Triggers and constraints are enabled on the standby database but they are not executed. For triggers and constraints on tables maintained through SQL apply operations, constraints are evaluated on the primary database and do not need to be re-evaluated on the logical standby database. The effects of the triggers executed on the primary database are logged and applied on the standby database. Triggers will

be fired and constraints will be evaluated on tables not maintained through SQL apply operations.

### 9.1.5 Skipping SQL Apply Operations on a Logical Standby Database

If only a subset of activity on a primary database is of interest on the standby database, use the DBMS_LOGSTDBY.SKIP procedure to define filters that prevent log apply services from issuing the SQL statements on the logical standby database. (See Section 4.1.4 for information about SQL statements that are skipped automatically.)

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

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

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

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

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

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

***Example 9–2 Skipping ALTER or CREATE TABLESPACE Statements***

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

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

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

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

Use the SKIP procedure with caution, particularly when skipping DDL statements. If a CREATE TABLE statement is skipped, for example, you must also skip any other DDL statements that refer to that table. Otherwise, these statements will fail and cause an exception. When this happens, the SQL apply services stop running and will need to be manually restarted.

### 9.1.6 Adding or Re-Creating Tables on a Logical Standby Database

Typically, you use table instantiation to re-create a table after an unrecoverable operation. You can also use the procedure to enable SQL apply operations on a table that was formerly skipped.

Before you can create a table, it must meet the requirements described in Section 4.1.4 and Section 4.1.5 that explain:

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

> **Note:** The DBMS_LOGSTDBY PL/SQL package does not support the BLOB datatype, even though BLOB datatypes are supported by logical standby databases.

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

1. Stop log apply services to stop applying SQL statements to the database.

2. Ensure no operations are being skipped by querying the DBA_LOGSTDBY_SKIP view.

   If any operations are being skipped, resume application of each operation that is currently being skipped by using the DBMS_LOGSTDBY.UNSKIP procedure. If multiple filters were created on the table, you will need to execute the procedure multiple times.

3. Re-create the table in the logical standby database using the DBMS_LOGSTDBY.INSTANTIATE_TABLE procedure. In addition to creating a table, this procedure also imports the data from the primary table using a database link. The link supplied to this procedure must have LOGSTDBY_ADMINISTRATOR role granted on the primary database.

4. Resume log apply services.

Before accessing data in the newly added table, you should archive the current redo log on the primary database, and ensure that it is applied to the logical standby database.

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

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

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

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

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

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

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

When the value returned by the DBA_LOGSTDBY_PROGRESS.APPLIED_SCN procedure is greater than the value selected from the query of the V$LOG view, the database is consistent and you can safely run reports again.

### 9.1.7 Viewing and Controlling Logical Standby Events

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

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

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

Errors that cause SQL apply operations to stop are always recorded in the events table (unless there is insufficient space in the system tablespace). These events are always put into the ALERT.LOG file as well, with the phrase 'LOGSTDBY event' included in the text. When querying the view, select the columns in order by EVENT_TIME, COMMIT_SCN, and CURRENT_SCN. This ordering ensures that a shutdown failure appears last in the view.

### 9.1.8 Viewing SQL Apply Operations Activity

SQL apply operations for logical standby databases use a collection of parallel execution servers and background processes to perform a number of different tasks. The V$LOGSTDBY view shows what each process is currently doing; the TYPE column describes the task being performed:

- The COORDINATOR process (LSP) is the background process that starts the other processes and schedules transactions.
- The READER process reads redo records from the archived redo logs.
- The PREPARER processes do the heavy computing required to convert the block changes into table changes.
- The BUILDER process assembles completed transactions.
- The ANALYZER process examines the records, possibly eliminating transactions and performing some dependency computation.
- The APPLIER processes generate and execute the completed SQL transactions.

When querying the V$LOGSTDBY view, pay special attention to the HIGH_SCN column. This is an activity indicator. As long as it is changing each time you query

the V$LOGSTDBY view, progress is being made. The STATUS column gives a text description of the current activity. For example:

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

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

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

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

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

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

NAME                                VALUE
----------------------------------- -----------------------------------
```

```
coordinator state                  APPLYING
transactions ready                 7821
transactions applied               7802
coordinator uptime                 73
```

This query shows how long SQL apply operations have been running and how many transactions have been applied in that time. It also shows how many transactions are available to be applied, indicating that more work is necessary.

## 9.1.9 Delaying the Application of Archived Redo Logs

Specifying an apply delay interval (in minutes) on the primary database is the same for both logical and physical standby databases (as described in Section 5.3.2.3, "Specifying a Time Lag for the Application of Redo Logs"). However, on a logical standby database, if the primary database is no longer available, you can cancel the apply delay interval by specifying the following PL/SQL command:

```
SQL> EXECUTE DBMS_LOGSTDBY.APPLY_UNSET('APPLY_DELAY');
```

## 9.1.10 Determining How Much Redo Log Data Was Applied

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

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

Logs with a NEXT_CHANGE# below READ_SCN are no longer needed. The information in those logs was applied or persistently stored in the database. The time values associated with these SCN values are only estimates based on log times. They are not meant to be accurate times of when those SCN values were written on the primary database.

You can see which logs were applied or were not applied by using the following query:

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

SQL> SELECT L.SEQUENCE#, L.FIRST_TIME,
```

```
  2    (CASE WHEN L.NEXT_CHANGE# < P.READ_SCN THEN 'YES'
  3          WHEN L.FIRST_CHANGE# < P.APPLIED_SCN THEN 'CURRENT'
  4          ELSE 'NO' END) APPLIED
  5  FROM DBA_LOGSTDBY_LOG L, DBA_LOGSTDBY_PROGRESS P
  6  ORDER BY SEQUENCE#;

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

19 rows selected.
```

## 9.1.11 Recovering from Errors

Logical standby databases maintain user tables, sequences, and jobs. To maintain other objects, you must reissue the DDL statements seen in the redo data stream. Tables in the SYS schema are never maintained, because only Oracle metadata is maintained in the SYS schema.

If a SQL apply operation fails, an error is recorded in the DBA_LOGSTDBY_EVENTS table. The following sections demonstrate how to recover from two such errors.

### 9.1.11.1 DDL Transactions Containing File Specifications

DDL statements are executed the same way on both the primary database and logical standby databases. If the underlying file structure is the same on both

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

If an error was caused by a DDL transaction that contained a file specification that does not match in the logical standby database environment, perform the following steps to fix the problem:

1. Use the DBMS_LOGSTDBY.GUARD_BYPASS_ON procedure to bypass the database guard so you can make modifications to the logical standby database:

   ```
   SQL> EXECUTE DBMS_LOGSTDBY.GUARD_BYPASS_ON;
   ```

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> EXECUTE DBMS_LOGSTDBY.GUARD_BYPASS_OFF;
   ```

3. Query the DBA_LOGSTDBY_EVENTS view to find the XIDUSN, XIDSLT, and XIDSQN values for the failed DDL, and provide the values to the DBMS_LOGSTDBY.SKIP_TRANSACTION procedure. The failed DDL statement will always be the last transaction. For example:

   ```
   SQL> SELECT XIDUSN, XIDSLT, XIDSQN FROM DBA_LOGSTDBY_EVENTS
     2> WHERE EVENT_TIME = (SELECT MAX(EVENT_TIME) FROM DBA_LOGSTDBY_EVENTS);
   SQL> EXECUTE DBMS_LOGSTDBY.SKIP_TRANSACTION( /*xidusn*/, /*xidslt*/,
   /*xidsqn*/);
   ```

4. Start log apply services on the logical standby database.

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

When log apply services restart, they will attempt to re-execute the transaction that failed. If you do not want to re-execute it, provide the values to the DBMS_LOGSTDBY.SKIP_TRANSACTION procedure (see step 3 for an example) to skip the transaction.

In some situations, the problem that caused the transaction to fail can be corrected and log apply services restarted without skipping the transaction. An example of this might be when available space is exhausted. The example shows log apply services stopping, how to correct the error, and then restart log apply services. For example:

```
SQL> SELECT * FROM DBA_LOGSTDBY_EVENTS;
EVENT_TIM CURRENT_SCN COMMIT_SCN     XIDUSN     XIDSLT     XIDSQN
--------- ---------- ---------- ---------- ---------- ----------
EVENT

--------------------------------------------------------------------------------
STATUS_CODE
-----------
STATUS

--------------------------------------------------------------------------------
30-JUL-02

      16111
ORA-16111: log mining and apply setting up

30-JUL-02      200240     200243          1          2     2213
create table bar (x number, y number) tablespace foo
      16204
ORA-16204: DDL successfully applied

30-JUL-02      200695     200735          1         11     2215
SCOTT.BAR (Oper=INSERT)
       1653
ORA-01653: unable to extend table SCOTT.BAR by %d in tablespace

30-JUL-02      200812     200864          1         11     2215
SCOTT.BAR (Oper=INSERT)
       1653
ORA-01653: unable to extend table SCOTT.BAR by %d in tablespace
```

In the example, the ORA-01653 message indicates that the tablespace was full and unable to extend itself. To correct the problem, add a new datafile to the tablespace. For example:

```
SQL> ALTER TABLESPACE t_table ADD DATAFILE 'dbs/t_db.f' SIZE 60M;
Tablespace altered.
```

Then, restart log apply services:

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

When log apply services restart, the transaction that failed will be re-executed and applied to the logical standby database.

#### 9.1.11.2 Recovering from DML Failures

Although the SKIP_TRANSACTION procedure can be very helpful, you should be cautious when using it to filter DML failures. Not only is the DML that is seen in the events table skipped, but so is all the DML associated with the transaction. Thus, multiple tables might be damaged by such an action.

DML failures usually indicate a problem with a specific table. For example, assume the failure is an out-of-storage error that you cannot resolve immediately. The following steps demonstrate one way to respond to this problem.

1. Bypass the table, but not the transaction, by adding the table to the skip list:

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

   From this point on, DML activity for the SCOTT.EMP table will not be applied. After you correct the storage problem, you can fix the table, provided that 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. The link supplied to this procedure must have LOGSTDBY_ADMINISTRATOR role granted on the primary database.

   ```
   SQL> ALTER DATABASE STOP LOGICAL STANDBY APPLY;
   SQL> EXECUTE DBMS_LOGSTDBY.INSTANTIATE_TABLE('SCOTT','EMP','PRIMARYDB');
   SQL> ALTER DATABASE START LOGICAL STANDBY APPLY;
   ```

3. Because table SCOTT.EMP will contain records as of when the INSTANTIATE_TABLE procedure was performed (in step 2), it is possible for the SCOTT.EMP table to contain records for a department not in the SCOTT.DEPT table.

### 9.1.12 Refreshing Materialized Views

Materialized views refreshed on the primary database are not automatically refreshed separately on a logical standby database. To refresh materialized views on the logical standby database, use the GUARD_BYPASS_ON and GUARD_BYPASS_OFF procedures of the DBMS_LOGSTDBY package. For example:

```
EXECUTE DBMS_LOGSTDBY.GUARD_BYPASS_ON;
EXECUTE DBMS_MVIEW.REFRESH ( 'BMVIEW', 'F', '',TRUE,FALSE,0,0,0,FALSE);
EXECUTE DBMS_LOGSTDBY.GUARD_BYPASS_OFF;
```

> **See Also:** *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about the DBMS_LOGSTDBY package

If you are using the DBMS_LOGSTDBY.APPLY_SET procedure but you are not using the default value of FULL for the TRANSACTION_CONSISTENCY parameter, you should stop SQL apply operations before refreshing materialized views on the logical standby database.

## 9.2 Tuning Logical Standby Databases

Take the following actions to increase system performance:

- On the primary database, if a table does not have a primary key or a unique index, then create a primary key RELY constraint. On the logical standby database, create an index on the columns that make up the primary key. The following query generates a list of tables with no index information that can be used by a logical standby database to apply to uniquely identify rows. By creating an index on the following tables, performance can be improved significantly.

  ```
  SQL> SELECT OWNER, TABLE_NAME FROM DBA_TABLES
    2> WHERE OWNER NOT IN('SYS','SYSTEM','OUTLN','DBSNMP')
    3> MINUS
    3> SELECT DISTINCT TABLE_OWNER, TABLE_NAME FROM DBA_INDEXES
    4> WHERE INDEX_TYPE NOT LIKE ('FUNCTION-BASED%')
    5> MINUS
    6> SELECT OWNER, TABLE_NAME FROM DBA_LOGSTDBY_UNSUPPORTED;
  ```

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

  ```
  SQL> EXECUTE DBMS_LOGSTDBY.GUARD_BYPASS_ON;
  SQL> CREATE INDEX EMPI ON EMP (EMPNO);
  SQL> EXECUTE DBMS_LOGSTDBY.GUARD_BYPASS_OFF;
  ```

  > **See Also:** Section 4.1.2 and *Oracle9i SQL Reference* for more information about RELY constraints

- Gather statistics for the cost-based optimizer (CBO) periodically on the logical standby database for objects, where the statistics become stale over time because of changing data volumes or changes in column values. 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/DDL operations on the primary are executed as a function of the workload. While the standby database is logically equivalent to the primary, SQL apply operations might execute the workload in a different way. This is why using the DBMS_STATS package on the logical standby database and the V$SYSSTAT view can be useful in determining which tables are consuming the most resources and table scan operations.

**See Also:** *Oracle9i Supplied PL/SQL Packages and Types Reference*

- Adjust the transaction consistency.

  Use the TRANSACTION_CONSISTENCY parameter of the DBMS_LOGSTDBY.APPLY_SET procedure to control how transactions are applied to the logical standby database. The default setting is FULL, which applies transactions to the logical standby database in the same order in which they were committed on the primary database.

  Specify one of the following values:

  - FULL

    Transactions are applied to the logical standby database in the exact order in which they were committed on the primary database. This option results in the lowest performance. This is the default parameter setting.

  - READ_ONLY

    Transactions are applied out of order from how they were committed on the primary database. The READ_ONLY option provides better performance than the FULL value, and SQL SELECT statements return read-consistent results. This is particularly beneficial when you are using the logical standby database to generate reports.

  - NONE

    Transactions are applied out of order from how they were committed on the primary database, and no attempt is made to provide read-consistent results. This results in the best performance of the three values. If applications that are reading the logical standby database make no assumptions about transaction order, this option works well.

**Note:** The READ_ONLY and NONE options should only be used when ALTER DATABASE GUARD ALL is set.

- Adjust the maximum number of parallel execution processes.

  Use the PARALLEL_MAX_SERVERS initialization parameter to adjust the maximum number of parallel execution processes and parallel recovery processes for an instance. The default value for this parameter is derived from the values of the CPU_COUNT, PARALLEL_AUTOMATIC_TUNING, and PARALLEL_ADAPTIVE_MULTI_USER initialization parameters. This parameter must not be set to a value less than 5 on a logical standby database.

  You can use the MAX_SERVERS parameter of the DBMS_LOGSTDBY.APPLY_SET procedure to limit the number of parallel servers used by log apply services. The default value of this parameter is set equal to the value of the PARALLEL_MAX_SERVERS initialization parameter. If you set this parameter explicitly, do not set it to a value less than 5 or greater than the value of the PARALLEL_MAX_SERVERS initialization parameter.

  Increasing the number of parallel execution processes and parallel recovery processes for an instance can speed up execution and recovery operations, but this improvement must be balanced against the consumption of additional system resources by the processes.

- Control memory usage on the logical standby database.

  You can use the MAX_SGA parameter of the DBMS_LOGSTDBY.APPLY_SET procedure to set the maximum amount of shared pool space used by log apply services for redo cache. By default, log apply services will use up to one quarter of the shared pool. Generally speaking, increasing the size of the shared pool or the amount of shared pool space used by log apply services will improve the performance of a logical standby database.

# 10

# Data Guard Scenarios

This chapter provides a collection of typical scenarios you might encounter while administering your Data Guard configuration. Each scenario is presented as a detailed step-by-step example that can be adapted to your specific environment. Table 10–1 lists each of the scenarios presented in this chapter.

***Table 10–1 Data Guard Scenarios***

| Reference | Scenario |
|---|---|
| Section 10.1 | Choosing the Best Available Standby Database for a Role Transition |
| Section 10.2 | Using a Physical Standby Database with a Time Lag |
| Section 10.3 | Switching Over to a Physical Standby Database That Has a Time Lag |
| Section 10.4 | Recovering from a Network Failure |
| Section 10.5 | Recovering After the NOLOGGING Clause Is Specified |

## 10.1 Choosing the Best Available Standby Database for a Role Transition

Every standby database is associated with one and only one primary database. A single primary database can, however, support multiple physical or logical standby databases. This scenario illustrates how to determine the information you need to choose the best available standby database for a failover or switchover operation.

For most role transitions (failovers or switchovers), if the configuration contains physical standby databases, Oracle recommends that you perform the role transition using the best available physical standby database. This is recommended because:

- A logical standby database might contain only a subset of the data present in the primary database.
- A role transition involving a logical standby database requires that any existing physical standby databases be re-created from a copy of the new primary database (after the role transition is complete) to continue to participate in the Data Guard configuration.

Because of these limitations, a logical standby database should be considered as the target for a role transition only in the the following special situations:

- The configuration contains only logical standby databases
- It is critical to fail over a standby database to the primary role as quickly as possible and the most current logical standby database in the configuration is significantly more current than the most current physical standby database in the configuration

Once you determine whether to use a physical or a logical standby database, the specific standby database you select as the target for the role transition is determined by how much of the recent primary database modifications are available at the standby location and by how much of these modifications were applied to the standby database. Because the primary database remains accessible during switchover operations, there will be no loss of data, and the choice of the standby database used during a switchover will only affect the time required to complete the switchover. For failovers, however, the choice of standby database might involve tradeoffs between additional risk of data loss and the time required to transition a standby database to the primary role.

### 10.1.1 Example: Best Physical Standby Database for a Failover Operation

In a disaster, the most critical task for the DBA is to determine if it is quicker and safer to repair the primary database or fail over to a standby database. When deciding that a failover operation is necessary and multiple physical standby databases are configured, you must choose which physical standby database is the best target for the failover operation. While there are many environmental factors that can affect which standby database represents the best choice, this scenario assumes these things to be equal for the purpose of emphasizing data loss assessment.

This scenario begins with a Data Guard configuration consisting of the HQ primary database and two physical standby databases, SAT and NYC. The HQ database is operating in maximum availability protection mode, and the standby databases are each configured with three standby redo logs.

**See Also:** Section 5.2 for more information about the maximum availability protection mode for physical standby databases

Table 10–2 provides information about the databases used in this scenario.

***Table 10–2 Identifiers for the Physical Standby Database Example***

| Identifier | HQ Database | SAT Database | NYC Database |
|---|---|---|---|
| Location | San Francisco | Seattle | New York City |
| Database name | HQ | HQ | HQ |
| Instance name | HQ | SAT | NYC |
| Initialization parameter file | hq_init.ora | sat_init.ora | nyc_init.ora |
| Control file | hq_cf1.f | sat_cf1.f | nyc_cf1 |
| Datafile | hq_db1.f | sat_db1.f | sat_db1.f |
| Online redo log file 1 | hq_log1.f | sat_log1.f | nyc_log1.f |
| Online redo log file 2 | hq_log2.f | sat_log2.f | nyc_log2.f |
| Standby redo log file 1 | hq_srl1.f | sat_srl1.f | nyc_srl1.f |
| Standby redo log file 2 | hq_srl2.f | sat_srl2.f | nyc_srl2.f |
| Standby redo log file 3 | hq_srl3.f | sat_srl3.f | nyc_srl3.f |
| Primary protection mode | Maximum availability | Not applicable | Not applicable |
| Standby protection mode | Not applicable | Maximum availability (synchronous) | Maximum performance (asynchronous) |
| Network service name (client defined) | hq_net | sat_net | nyc_net |
| Listener | hq_listener | sat_listener | nyc_listener |

**Note:** The New York city database is operating in maximum performance mode because sending redo data synchronously from HQ to NYC might impact the primary database performance during peak workload periods. However, the New York City standby database is still considered a viable candidate for failover operations because it uses standby redo logs.

Assume that an event occurs in San Francisco where the primary site is located, and the primary site is damaged in such a way that it cannot be repaired in a timely manner. You must fail over to one of the standby databases. You cannot assume that the DBA who set up the multiple standby database configuration is available to decide to which standby database to fail over. Therefore, it is imperative to have a disaster recovery plan at each standby site, as well as at the primary site. Each member of the disaster recovery team needs to know about the disaster recovery plan and be aware of the procedures to follow. This scenario identifies the information you need when deciding which standby database should be the target of the failover operation.

One method of conveying information to the disaster recovery team is to include a ReadMe file at each standby site. This ReadMe file is created and maintained by the DBA and should describe how to:

- Log on to the local database server as a DBA
- Log on to each system where the standby databases are located
- Get instructions for going through firewalls, because there might be firewalls between systems
- Log on to other database servers as a DBA
- Identify the most up-to-date standby database
- Perform the standby database failover operation
- Configure network settings to ensure that client applications access the new primary database, instead of the original primary database

**See Also:** Appendix E for a sample ReadMe file

When choosing a standby database, there are two critical considerations: which standby database received the most recent redo data and which standby database has applied the most redo logs.

Follow these steps to determine which standby database is the best candidate for failover when only physical standby databases are in the configuration. Always start with the standby database providing the highest protection level. In this scenario, the Seattle standby database provides the highest protection level because it is operating in maximum availability protection level.

### Step 1 Connect to the SAT physical standby database.

Issue a SQL statement such as the following:

```
SQL> CONNECT SYS/CHANGE_ON_INSTALL AS SYSDBA;
```

### Step 2 Determine how much current redo data is available in the redo log.

Query the columns in the V$MANAGED_STANDBY view, as shown:

```
SQL> SELECT THREAD#, SEQUENCE#, BLOCK#, BLOCKS
     2> FROM V$MANAGED_STANDBY WHERE STATUS='RECEIVING';
   THREAD#  SEQUENCE#     BLOCK#     BLOCKS
---------- ---------- ---------- ----------
         1         14        234         16
```

This standby database received 249 blocks of redo data from the primary database. To compute the number of blocks received, add the BLOCKS column value to the BLOCK# column value, and subtract 1 (because block number 234 is included in the 16 blocks received).

> **Note:** Depending on how long the primary database has been unavailable, the previous query might not return any selected rows because the RFS process might detect the network disconnection and terminate itself. If this occurs, it is always best to select a standby database that is configured to receive the redo data in a synchronous manner.

### Step 3 Obtain a list of the archived redo logs that were applied or are currently pending application to the SAT database.

Query the V$ARCHIVED_LOG view:

```
SQL> SELECT SUBSTR(NAME,1,25) FILE_NAME, SEQUENCE#, APPLIED
  2> FROM V$ARCVHIVED_LOG ORDER BY SEQUENCE#;
```

```
FILE_NAME                  SEQUENCE# APP
-------------------------- ---------- ---
/oracle/dbs/hq_sat_2.log            2 YES
/oracle/dbs/hq_sat_3.log            3 YES
/oracle/dbs/hq_sat_4.log            4 YES
/oracle/dbs/hq_sat_5.log            5 YES
/oracle/dbs/hq_sat_6.log            6 YES
/oracle/dbs/hq_sat_7.log            7 YES
/oracle/dbs/hq_sat_8.log            8 YES
/oracle/dbs/hq_sat_9.log            9 YES
/oracle/dbs/hq_sat_10.log          10 YES
/oracle/dbs/hq_sat_11.log          11 YES
/oracle/dbs/hq_sat_13.log          13  NO
```

This output indicates that archived redo log 11 was completely applied to the standby database. (The line for log 11 in the example output is in bold typeface to assist you in reading the output. The actual output will not display bolding.)

Also, notice the gap in the sequence numbers in the SEQUENCE# column. In the example, the gap indicates that the SAT standby database is missing archived redo log number 12.

### Step 4 Connect to the NYC database to determine if it is more recent than the SAT standby database.

Issue a SQL statement such as the following:

```
SQL> CONNECT SYS/CHANGE_ON_INSTALL AS SYSDBA;
```

### Step 5 Determine how much current redo data is available in the redo log.

Query the columns in the V$MANAGED_STANDBY view as shown:

```
SQL> SELECT THREAD#, SEQUENCE#, BLOCK#, BLOCKS
     2> FROM V$MANAGED_STANDBY WHERE STATUS='RECEIVING';
   THREAD#  SEQUENCE#     BLOCK#     BLOCKS
---------- ---------- ---------- ----------
         1         14        157         93
```

This standby database has also received 249 blocks of redo information from the primary database. To compute the number of blocks received, add the BLOCKS column value to the BLOCK# column value, and subtract 1 (because block number 157 is included in the 93 blocks received).

**Step 6 Obtain a list of the archived redo logs that were applied or are currently pending application to the NYC database.**

Query the V$ARCHIVED_LOG view:

```
SQL> SELECT SUBSTR(NAME,1,25) FILE_NAME, SEQUENCE#, APPLIED
  2> FROM V$ARCVHIVED_LOG ORDER BY SEQUENCE#;
FILE_NAME                 SEQUENCE# APP
------------------------- ---------- ---
/oracle/dbs/hq_nyc_2.log           2 YES
/oracle/dbs/hq_nyc_3.log           3 YES
/oracle/dbs/hq_nyc_4.log           4 YES
/oracle/dbs/hq_nyc_5.log           5 YES
/oracle/dbs/hq_nyc_6.log           6 YES
/oracle/dbs/hq_nyc_7.log           7 YES
/oracle/dbs/hq_nyc_8.log           8  NO
/oracle/dbs/hq_nyc_9.log           9  NO
/oracle/dbs/hq_nyc_10.log         10  NO
/oracle/dbs/hq_nyc_11.log         11  NO
/oracle/dbs/hq_nyc_12.log         12  NO
/oracle/dbs/hq_nyc_13.log         13  NO
```

This output indicates that archived redo log 7 was completely applied to the standby database. (The line for log 7 in the example output is in bold typeface to assist you in reading the output. The actual output will not display bolding.)

More redo data was received at this location, but less was applied to the standby database.

**Step 7 Choose the best target standby database.**

In most cases, the physical standby database you choose as a failover target should provide a balance between risk of data loss and time required to perform the role transition. As you analyze this information to make a decision about the best failover candidate in this scenario, consider the following:

- For minimal risk of data loss during a failover operation, you should choose the NYC database as the best target standby database because steps 5 and 6 revealed that the NYC site has the most recoverable redo logs.
- For minimal primary database downtime during the failover operation, you should choose the SAT database as the best target standby database. This database is a more appropriate candidate because the queries in steps 2 through 6 reveal that the SAT database applied 5 archived redo logs more than the NYC database. However, if it is not possible to obtain and apply a copy of the missing archived log (archived redo log 12 in the example), then you will not be

able to make the SAT database as current as you can the NYC database. Therefore, you will lose the unapplied data (logs 12, 13, and part of log 14 in the example).

Based on your business requirements, choose the best target standby database.

**Step 8 Bring the selected standby database to its most current state.**

**If you chose the SAT database as the best target based on your business requirements, perform the following steps:**

1. Manually retrieve any missing archived redo logs using an operating system utility. (This example uses the UNIX `cp` command). In this case, the SAT database is missing archived redo log 12. Because the NYC database received this archived redo log, you can copy it from the NYC database to the SAT database, as follows:

   ```
   % cp /net/nyc/oracle/dbs/hq_nyc_12.log /net/sat/oracle/dbs/hq_sat_12.log
   ```

2. Determine if a partial archived redo log exists for the next sequence number. In this example, the next sequence number should be 14. The following UNIX command searches the directory on the SAT database for the presence of an archived redo log named `hq_sat_14.log`:

   ```
   % ls -l /net/sat/oracle/dbs/hq_sat_14.log
   /net/sat/oracle/dbs/hq_sat_14.log: No such file or directory
   ```

   Because the SAT standby database is using standby redo logs, there should not be any partial archived redo logs.

3. Register the retrieved archived redo log. (There is no need to stop the log apply services).

   ```
   SQL> ALTER DATABASE REGISTER PHYSICAL LOGFILE '/oracle/dbs/hq_sat_12.log';
   ```

4. Query the `V$ARCHIVED_LOG` view again to make sure the archived redo logs were successfully applied:

   ```
   SQL> SELECT SUBSTR(NAME,1,25) FILE_NAME, SEQUENCE#, APPLIED
     2> FROM V$ARCVHIVED_LOG ORDER BY SEQUENCE#;

   FILE_NAME                 SEQUENCE# APP
   ------------------------- ---------- ---
   /oracle/dbs/hq_sat_2.log           2 YES
   /oracle/dbs/hq_sat_3.log           3 YES
   /oracle/dbs/hq_sat_4.log           4 YES
   ```

```
/oracle/dbs/hq_sat_5.log           5 YES
/oracle/dbs/hq_sat_6.log           6 YES
/oracle/dbs/hq_sat_7.log           7 YES
/oracle/dbs/hq_sat_8.log           8 YES
/oracle/dbs/hq_sat_9.log           9 YES
/oracle/dbs/hq_sat_10.log         10 YES
/oracle/dbs/hq_sat_11.log         11 YES
/oracle/dbs/hq_sat_12.log         12 YES
/oracle/dbs/hq_sat_13.log         13 YES
```

### If you chose the NYC database as the best target based on your business requirements, perform the following steps:

1. Determine if a partial archived redo log exists for the next sequence number. The following UNIX command searches the directory on the NYC database for the presence of an archived redo log named with the next sequence named (hq_nyc_14):

   ```
   % ls -l /net/nyc/oracle/dbs/hq_nyc_14.log
   /net/nyc/oracle/dbs/hq_nyc_14.log: No such file or directory
   ```

   Because the NYC standby database is using standby redo logs, there should not be any partial archived redo logs.

2. Start log apply services to apply the most current log:

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

3. Query the V$ARCHIVED_LOG view again to make sure the archived redo logs were successfully applied:

   ```
   SQL> SELECT SUBSTR(NAME,1,25) FILE_NAME, SEQUENCE#, APPLIED
     2> FROM V$ARCVHIVED_LOG ORDER BY SEQUENCE#;
   FILE_NAME                  SEQUENCE# APP
   ------------------------- ---------- ---
   /oracle/dbs/hq_nyc_2.log           2 YES
   /oracle/dbs/hq_nyc_3.log           3 YES
   /oracle/dbs/hq_nyc_4.log           4 YES
   /oracle/dbs/hq_nyc_5.log           5 YES
   /oracle/dbs/hq_nyc_6.log           6 YES
   /oracle/dbs/hq_nyc_7.log           7 YES
   /oracle/dbs/hq_nyc_8.log           8 YES
   /oracle/dbs/hq_nyc_9.log           9 YES
   /oracle/dbs/hq_nyc_10.log         10 YES
   ```

```
/oracle/dbs/hq_nyc_11.log         11 YES
/oracle/dbs/hq_nyc_12.log         12  NO
/oracle/dbs/hq_nyc_13.log         13  NO
```

Applying the archived redo logs might take some time to complete. Therefore, you must wait until all archived redo logs are designated as applied, as shown:

```
SQL> SELECT SUBSTR(NAME,1,25) FILE_NAME, SEQUENCE#, APPLIED
  2> FROM V$ARCVHIVED_LOG ORDER BY SEQUENCE#;

FILE_NAME                  SEQUENCE# APP
------------------------- ---------- ---
/oracle/dbs/hq_nyc_2.log           2 YES
/oracle/dbs/hq_nyc_3.log           3 YES
/oracle/dbs/hq_nyc_4.log           4 YES
/oracle/dbs/hq_nyc_5.log           5 YES
/oracle/dbs/hq_nyc_6.log           6 YES
/oracle/dbs/hq_nyc_7.log           7 YES
/oracle/dbs/hq_nyc_8.log           8 YES
/oracle/dbs/hq_nyc_9.log           9 YES
/oracle/dbs/hq_nyc_10.log         10 YES
/oracle/dbs/hq_nyc_11.log         11 YES
/oracle/dbs/hq_nyc_12.log         12 YES
/oracle/dbs/hq_nyc_13.log         13 YES
```

**Step 9 Perform the failover operation.**

You are now ready to stop log apply services and fail over the selected physical standby database to the primary role.

---

**See Also:** Section 7.2.2 for additional information about how to perform a failover operation to a physical standby database

---

## 10.1.2 Example: Best Logical Standby Database for a Failover Operation

In a disaster when only logical standby databases are available, the critical task is to determine which logical standby database is the best target for the failover operation. While there are many environmental factors that can affect which is the best target standby database, this scenario assumes these things to be equal for the purpose of emphasizing data loss assessment.

**See Also:** Section 5.2 for more information about the maximum availability protection mode for logical standby databases

This scenario starts out with a Data Guard configuration consisting of the HQ primary database and two logical standby databases, SAT and NYC. Table 10–3 provides information about each of these databases.

***Table 10–3 Identifiers for Logical Standby Database Example***

| Identifier | HQ Database | SAT Database | NYC Database |
|---|---|---|---|
| Location | San Francisco | Seattle | New York City |
| Database name | HQ | SAT | NYC |
| Instance name | HQ | SAT | NYC |
| Initialization parameter file | hq_init.ora | sat_init.ora | nyc_init.ora |
| Control file | hq_cf1.f | sat_cf1.f | nyc_cf1.f |
| Datafile | hq_db1.f | sat_db1.f | nyc_db1.f |
| Online redo log file 1 | hq_log1.f | sat_log1.f | nyc_log1.f |
| Online redo log file 2 | hq_log2.f | sat_log2.f | nyc_log2.f |
| Database link (client-defined) | hq_link | sat_link | nyc_link |
| Network service name (client-defined) | hq_net | sat_net | nyc_net |
| Listener | hq_listener | sat_listener | nyc_listener |

Follow these steps to determine which standby database is the best candidate for failover when only logical standby databases are in the configuration:

### Step 1 Connect to the SAT logical standby database.

Issue a SQL statement such as the following:

```
SQL> CONNECT SYS/CHANGE_ON_INSTALL AS SYSDBA;
```

### Step 2 Determine the highest applied SCN and highest (newest) applicable SCN on the SAT database.

Query the following columns in the DBA_LOGSTDBY_PROGRESS view:

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

APPLIED_SCN NEWEST_SCN
----------- ----------
     144059     144059
```

**Step 3 Obtain a list of the archived redo logs that were applied or are currently pending application to the SAT database.**

Query the DBA_LOGSTDBY_LOG view:

```
SQL> SELECT SUBSTR(FILE_NAME,1,25) FILE_NAME, SUBSTR(SEQUENCE#,1,4) "SEQ#",
  2> FIRST_CHANGE#, NEXT_CHANGE#, TO_CHAR(TIMESTAMP, 'HH:MI:SS') TIMESTAMP,
  3> DICT_BEGIN BEG, DICT_END END, SUBSTR(THREAD#,1,4) "THR#"
  4> FROM DBA_LOGSTDBY_LOG ORDER BY SEQUENCE#;

FILE_NAME                 SEQ# FIRST_CHANGE# NEXT_CHANGE# TIMESTAM BEG END THR#
------------------------- ---- ------------- ------------ -------- --- --- ----
/oracle/dbs/hq_sat_2.log  2           101579       101588 11:02:57 NO  NO  1
/oracle/dbs/hq_sat_3.log  3           101588       142065 11:02:01 NO  NO  1
/oracle/dbs/hq_sat_4.log  4           142065       142307 11:02:09 NO  NO  1
/oracle/dbs/hq_sat_5.log  5           142307       142739 11:02:47 YES YES 1
/oracle/dbs/hq_sat_6.log  6           142739       143973 12:02:09 NO  NO  1
/oracle/dbs/hq_sat_7.log  7           143973       144042 01:02:00 NO  NO  1
/oracle/dbs/hq_sat_8.log  8           144042       144051 01:02:00 NO  NO  1
/oracle/dbs/hq_sat_9.log  9           144051       144054 01:02:15 NO  NO  1
/oracle/dbs/hq_sat_10.log 10          144054       144057 01:02:20 NO  NO  1
/oracle/dbs/hq_sat_11.log 11          144057       144060 01:02:25 NO  NO  1
/oracle/dbs/hq_sat_13.log 13          144089       144147 01:02:40 NO  NO  1
```

Notice that for log 11, the SCN of 144059 (recorded in step 2) is between the FIRST_CHANGE# column value of 144057 and the NEXT_CHANGE# column value of 144060. This indicates that log 11 is currently being applied. (The line for log 11 in the example output is in bold typeface to assist you in reading the output. The actual output will not display bolding.) Also notice the gap in the sequence numbers in the SEQ# column; in the example, the gap indicates that SAT database is missing archived redo log 12.

**Step 4 Connect to the NYC database.**

Issue a SQL statement such as the following:

```
SQL> CONNECT SYS/CHANGE_ON_INSTALL AS SYSDBA;
```

**Step 5 Determine the highest applied SCN and highest applicable SCN on the NYC database.**

Query the following columns in the DBA_LOGSTDBY_PROGRESS view:

```
SQL> SELECT APPLIED_SCN, NEWEST_SCN FROM DBA_LOGSTDBY_PROGRESS;
APPLIED_SCN NEWEST_SCN
----------- ----------
     143970     144146
```

### Step 6 Obtain a list of the logs that were processed or are currently pending processing on the NYC database.

Issue a SQL statement such as the following:

```
SQL> SELECT SUBSTR(FILE_NAME,1,25) FILE_NAME, SUBSTR(SEQUENCE#,1,4) "SEQ#",
  2> FIRST_CHANGE#, NEXT_CHANGE#, TO_CHAR(TIMESTAMP, 'HH:MI:SS')  TIMESTAMP,
  3> DICT_BEGIN BEG, DICT_END END, SUBSTR(THREAD#,1,4) "THR#"
  4> FROM DBA_LOGSTDBY_LOG ORDER BY SEQUENCE#;

FILE_NAME                 SEQ# FIRST_CHANGE# NEXT_CHANGE# TIMESTAM BEG END THR#
------------------------- ---- ------------- ------------ -------- --- --- ----
/oracle/dbs/hq_nyc_2.log  2           101579       101588 11:02:58 NO  NO  1
/oracle/dbs/hq_nyc_3.log  3           101588       142065 11:02:02 NO  NO  1
/oracle/dbs/hq_nyc_4.log  4           142065       142307 11:02:10 NO  NO  1
/oracle/dbs/hq_nyc_5.log  5           142307       142739 11:02:48 YES YES 1
/oracle/dbs/hq_nyc_6.log  6           142739       143973 12:02:10 NO  NO  1
/oracle/dbs/hq_nyc_7.log  7           143973       144042 01:02:11 NO  NO  1
/oracle/dbs/hq_nyc_8.log  8           144042       144051 01:02:01 NO  NO  1
/oracle/dbs/hq_nyc_9.log  9           144051       144054 01:02:16 NO  NO  1
/oracle/dbs/hq_nyc_10.log 10          144054       144057 01:02:21 NO  NO  1
/oracle/dbs/hq_nyc_11.log 11          144057       144060 01:02:26 NO  NO  1
/oracle/dbs/hq_nyc_12.log 12          144060       144089 01:02:30 NO  NO  1
/oracle/dbs/hq_nyc_13.log 13          144089       144147 01:02:41 NO  NO  1
```

Notice that for log 6, the SCN of 143970 (recorded in step 5) is between the FIRST_CHANGE# column value of 142739 and the NEXT_CHANGE# column value of 143973. This indicates that log 6 is currently being applied. (The line for log 6 in the example output is in bold typeface to assist you in reading the output. The actual output will not display bolding.) Also, notice that there are no gaps in the sequence of logs that remain to be processed.

### Step 7 Choose the best target standby database.

In most cases, the logical standby database you choose as a failover target should provide a balance between risk of data loss and time required to perform the role transition. As you analyze this information to make a decision about the best failover candidate in this scenario, consider the following:

- For minimal risk of data loss during a failover operation, you should choose the NYC database as the best target standby database because steps 5 and 6 revealed that the NYC site has the most recoverable redo logs.
- For minimal primary database downtime during the failover operation, you should choose the SAT database as the best target standby database. This database is a more appropriate candidate because the queries in steps 2 through

6 reveal that the SAT database applied 5 archived redo logs more than the NYC database (even though there was only a 1-second delay (lag) in the receipt of archived redo logs by the NYC database). However, if it is not possible to obtain and apply a copy of the missing archived log (log 12 in the example), then you will not be able to make the SAT database as current as you can the NYC database. Therefore, you will lose the unrecovered data (logs 12, 13, and part of log 14 in the example).

Based on your business requirements, choose the best target standby database.

**Step 8 Bring the selected standby database to its most current state.**

### If you chose the SAT database as the best target based on your business requirements, perform the following steps:

1. Manually retrieve any missing archived redo logs using an operating system utility. In this case, the SAT database is missing archived redo log 12. Because the NYC database received this archived redo log, you can copy it from the NYC database to the SAT database, as follows:

   ```
   %cp /net/nyc/oracle/dbs/hq_nyc_12.log
   /net/sat/oracle/dbs/hq_sat_12.log
   ```

2. Determine if a partial archived redo log exists for the next sequence number. In this example, the next sequence number should be 14. The following UNIX command shows the directory on the SAT database, looking for the presence of an archived redo log named `hq_sat_14.log`:

   ```
   %ls -l /net/sat/oracle/dbs/hq_sat_14.log
   -rw-rw----   1 oracle    dbs  333280 Feb 12  1:03 hq_sat_14.log
   ```

3. Stop log apply services and register both the retrieved archived redo log and the partial archived redo log:

   ```
   SQL> ALTER DATABASE STOP LOGICAL STANDBY APPLY;
   SQL> ALTER DATABASE REGISTER LOGICAL LOGFILE '/oracle/dbs/hq_sat_12.log';
   SQL> ALTER DATABASE REGISTER LOGICAL LOGFILE '/oracle/dbs/hq_sat_14.log';
   ```

4. Start log apply services to apply the most current log:

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

5. Determine the highest applied SCN on the SAT database by querying the `DBA_LOGSTDBY_PROGRESS` view to see if the value of the `APPLIED_SCN` column is equal to the value of the `NEWEST_SCN` column:

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

APPLIED_SCN NEWEST_SCN
----------- ----------
     144205     144205
```

Because the SCN values match, you can be assured that there is no longer a delay (lag) between the current log on the primary database and the last log applied to the SAT database.

### If you chose the NYC database as the best target based on your business requirements, perform the following steps:

1. Determine if a partial archived redo log exists for the next sequence number. In this example, the next sequence number should be 14. The following UNIX command shows the directory on the NYC database, looking for the presence of an archived redo log named hq_nyc_14:

   ```
   %ls -l /net/nyc/oracle/dbs/hq_nyc_14.log
   -rw-rw----   1 oracle    dbs  333330 Feb 12  1:03 hq_nyc_14.log
   ```

2. Register the partial archive redo log on the NYC database:

   ```
   SQL> ALTER DATABASE STOP LOGICAL STANDBY APPLY;
   SQL> ALTER DATABASE REGISTER LOGICAL LOGFILE '/oracle/dbs/hq_nyc_14.log';
   ```

3. Start log apply services to apply the most current log:

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

4. Determine the highest applied SCN on the NYC database by querying the DBA_LOGSTDBY_PROGRESS view to see if the value of the APPLIED_SCN column is equal to the value of the NEWEST_SCN column:

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

   APPLIED_SCN NEWEST_SCN
   ----------- ----------
        144205     144205
   ```

   Because the SCN values match, you can sure there is no longer a delay (lag) between the current log on the primary database and the last log received and applied by the NYC database.

**Step 9 Perform the failover.**

You are now ready to stop log apply services and fail over the selected logical standby database to the primary role.

> **See Also:** Section 7.3.2 for additional information on how to perform the failover operation

## 10.2 Using a Physical Standby Database with a Time Lag

By default, when the standby database is performing managed recovery, it automatically applies redo logs when they arrive from the primary database. But in some cases, you might not want the logs to be applied immediately, because you want to create a time lag between the archiving of a redo log at the primary site and the application of the log at the standby site. A time lag can protect against the transfer of corrupted or erroneous data from the primary site to the standby site.

For example, suppose you run a batch job every night on the primary database. Unfortunately, you accidently ran the batch job twice and you did not realize the mistake until the batch job completed for the second time. Ideally, you need to roll back the database to the point in time before the batch job began. A primary database that has a standby database with a time lag could help you to recover. You could fail over the standby database with the time lag and use it as the new primary database.

To create a standby database with a time lag, use the `DELAY` attribute of the `LOG_ARCHIVE_DEST_`*n* initialization parameter in the primary database initialization parameter file. The archived redo logs are still automatically copied from the primary site to the standby site, but the logs are not immediately applied to the standby database. The logs are applied when the specified time interval expires.

This scenario uses a 4-hour time lag and covers the following topics:

- Establishing a Time Lag on a Physical Standby Database
- Failing Over to a Physical Standby Database with a Time Lag
- Switching Over to a Physical Standby Database That Has a Time Lag

Readers of this scenario are assumed to be familiar with the procedures for creating a typical standby database. The details were omitted from the steps outlined in this scenario.

> **See Also:** Chapter 3 for details about a creating physical databases

### 10.2.1 Establishing a Time Lag on a Physical Standby Database

To create a physical standby database with a time lag, modify the LOG_ARCHIVE_DEST_*n* initialization parameter on the primary database to set a delay for the standby database. The following is an example of how to add a 4-hour delay to the LOG_ARCHIVE_DEST_*n* initialization parameter:

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_2='SERVICE=stdby DELAY=240';
```

The DELAY attribute indicates that the archived redo logs at the standby site are not available for recovery until the 4-hour time interval has expired. The time interval (expressed in minutes) starts when the archived redo logs are successfully transmitted to the standby site. The redo information is still sent to the standby database and written to the disk as normal.

> **See Also:** *Oracle9i Supplied PL/SQL Packages and Types Reference* for a description of using the DBMS_LOGSTDBY package to establish a time lag on a logical standby database.

### 10.2.2 Failing Over to a Physical Standby Database with a Time Lag

A standby database configured to delay application of redo logs can be used to recover from user errors or data corruptions on the primary database. In most cases, you can query the time-delayed standby database to retrieve the data needed to repair the primary database (for example, to recover the contents of a mistakenly dropped table). In cases where the damage to the primary database is unknown or when the time required to repair the primary database is prohibitive, you can also consider failing over to a time-delayed standby database.

Assume that a backup file was inadvertently applied twice to the primary database and that the time required to repair the primary database is prohibitive. You choose to fail over to a physical standby database for which redo log application is delayed. By doing so, you transition the standby database to the primary role at a point before the problem occurred, but you will likely incur some data loss. The following steps illustrate the process:

1. Initiate the failover operation by issuing the appropriate SQL statements on the time-delayed physical standby database:

   ```
   SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE CANCEL;
   SQL> ALTER DATABASE ACTIVATE PHYSICAL STANDBY DATABASE SKIP STANDBY LOGFILE;
   SQL> SHUTDOWN IMMEDIATE;
   SQL> STARTUP
   ```

The ACTIVATE statement immediately transitions the standby database to the primary role and makes no attempt to apply any additional redo data that might exist at the standby location. When using this command, you must carefully balance the cost of data loss at the standby location against the potentially extended period of downtime required to fully repair the primary database.

2. Re-create all other standby databases in the configuration from a copy of this new primary database.

## 10.3 Switching Over to a Physical Standby Database That Has a Time Lag

All the redo logs are shipped to the standby site as they become available. Therefore, even when a time delay is specified for a standby database, you can make the standby database current by overriding the delay using the SQL ALTER DATABASE RECOVER MANAGED STANDBY statement.

> **Note:** To recover from a logical error, you must perform a failover operation instead of a switchover operation.

The following steps demonstrate how to perform a switchover to a time-delayed physical standby database that bypasses a time lag. For the purposes of this example, assume that the primary database is located in New York, and the standby database is located in Boston.

### Step 1 Apply all of the archived redo logs to the original (time delayed) standby database bypassing the lag.

Switchover will not begin until the standby database applies all of the redo logs. By lifting the delay, you allow the standby database to proceed without waiting for the apply operation to finish.

Issue the following SQL statement to lift the delay:

```
SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE NODELAY
  2> DISCONNECT FROM SESSION THROUGH LAST SWITCHOVER;
```

### Step 2 Stop read or update activity on the primary and standby databases.

You must have exclusive database access before beginning a switchover operation. Ask users to log off the primary and standby databases or query the V$SESSION

view to identify users that are connected to the databases and close all open sessions except the SQL*Plus session from which you are going to execute the switchover statement.

> **See Also:** *Oracle9i Database Administrator's Guide* for more information on managing users

### Step 3 Switch the primary database over to the physical standby role.

On the primary database (in New York), execute the following statement:

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

This statement does the following:

- Closes the primary database, terminating any active sessions
- Archives any unarchived logs and applies them to the standby database (in Boston)
- Adds an end-of-redo marker to the header of the last log being archived
- Creates a backup of the current control file
- Converts the current control file into a standby control file

### Step 4 Shut down and start up the former primary instance without mounting the database.

Execute the following statement on the former primary database (in New York):

```
SQL> SHUTDOWN NORMAL;
SQL> STARTUP NOMOUNT;
```

### Step 5 Mount the former primary database in the physical standby database role.

Execute the following statement to mount the former primary database (in New York) as a physical standby database:

```
SQL> ALTER DATABASE MOUNT STANDBY DATABASE;
```

### Step 6 Switch the original standby database to the primary role.

Issue the following SQL statement:

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

**Step 7 Shut down and restart the new primary database instance.**

Issue the following SQL statements:

```
SQL> SHUTDOWN;
SQL> STARTUP PFILE=Failover.ora;
```

## 10.4 Recovering from a Network Failure

The following steps describe how to recover after a network failure.

**Step 1 Identify the network failure.**

The V$ARCHIVE_DEST view contains the network error and identifies which standby database cannot be reached. On the primary database, execute the following SQL statement for the archived log destination that experienced the network failure. For example:

```
SQL> SELECT DEST_ID, STATUS, ERROR FROM V$ARCHIVE_DEST WHERE DEST_ID = 2;

DEST_ID    STATUS    ERROR
---------- --------- -----------------------------------------------------
         2  ERROR     ORA-12224: TNS:no listener
```

The query results show there are errors archiving to the standby database, and the cause of the error is TNS:no listener. You should check whether or not the listener on the standby site is started. If the listener is stopped, then start it.

**Step 2 Prevent the primary database from stalling.**

If you cannot solve the network problem quickly, and if the standby database is specified as a mandatory destination, try to prevent the database from stalling by doing one of the following:

- Defer archiving to the mandatory destination:

  ```
  SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2 = DEFER;
  ```

  When the network problem is resolved, you can enable the archive destination again:

  ```
  SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2 = ENABLE;
  ```

- Change the archive destination from mandatory to optional:

  ```
  SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_2 = 'SERVICE=standby1
    2> OPTIONAL REOPEN=60';
  ```

When the network problem is resolved, you can change the archive destination from optional back to mandatory:

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_2 = 'SERVICE=standby1
  2> MANDATORY REOPEN=60';
```

**Step 3 Archive the current redo log.**

On the primary database, archive the current redo log:

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

When the network is back up again, log apply services can detect and resolve the archive gaps automatically when the physical standby database resumes managed recovery operations.

## 10.5 Recovering After the NOLOGGING Clause Is Specified

In some SQL statements, the user has the option of specifying the NOLOGGING clause, which indicates that the database operation is not logged in the redo log. Even though the user specifies the clause, a redo log record is still written to the redo log. However, there is no data associated with this record. This can result in log application or data access errors at the standby site and manual recovery might be required to resume log apply operations.

---

**Note:** To avoid these problems, Oracle Corporation recommends that you always specify the FORCE LOGGING clause in the CREATE DATABASE or ALTER DATABASE statements. See the *Oracle9i Database Administrator's Guide.*

---

### 10.5.1 Recovery Steps for Logical Standby Databases

For logical standby databases, when SQL apply operations encounter a redo log record for an operation performed with the NOLOGGING clause, it skips over the record and continues applying changes from later records. Later, if an attempt is made to access one of the records that were updated with NOLOGGING in effect, the following error is returned: ORA-01403 no data found

To recover after the NOLOGGING clause is specified, re-create one or more tables from the primary database, as described in Section 9.1.6.

**Note:** In general, use of the NOLOGGING clause is not recommended. Optionally, if you know in advance that operations using the NOLOGGING clause will be performed on certain tables in the primary database, you might want to prevent the application of SQL statements associated with these tables to the logical standby database by using the DBMS_LOGSTDBY.SKIP procedure.

## 10.5.2 Recovery Steps for Physical Standby Databases

When the redo log is copied to the standby site and applied to the physical standby database, a portion of the datafile is unusable and is marked as being unrecoverable. When you either fail over to the physical standby database, or open the standby database for read-only access, and attempt to read the range of blocks that are marked as UNRECOVERABLE, you will see error messages similar to the following:

```
ORA-01578: ORACLE data block corrupted (file # 1, block # 2521)
ORA-01110: data file 1: '/oracle/dbs/stdby/tbs_1.dbf'
ORA-26040: Data block was loaded using the NOLOGGING option
```

To recover after the NOLOGGING clause is specified, you need to copy the datafile that contains the unjournaled data from the primary site to the physical standby site. Perform the following steps:

### Step 1 Determine which datafiles should be copied.

Follow these steps:

1. Query the primary database:

```
SQL> SELECT NAME, UNRECOVERABLE_CHANGE# FROM V$DATAFILE;
NAME                                               UNRECOVERABLE
-------------------------------------------------- -------------
/oracle/dbs/tbs_1.dbf                                       5216
/oracle/dbs/tbs_2.dbf                                          0
/oracle/dbs/tbs_3.dbf                                          0
/oracle/dbs/tbs_4.dbf                                          0
4 rows selected.
```

2. Query the standby database:

```
SQL> SELECT NAME, UNRECOVERABLE_CHANGE# FROM V$DATAFILE;
NAME                                               UNRECOVERABLE
-------------------------------------------------- -------------
```

```
/oracle/dbs/stdby/tbs_1.dbf                                    5186
/oracle/dbs/stdby/tbs_2.dbf                                       0
/oracle/dbs/stdby/tbs_3.dbf                                       0
/oracle/dbs/stdby/tbs_4.dbf                                       0
4 rows selected.
```

3. Compare the query results of the primary and standby databases.

   Compare the value of the UNRECOVERABLE_CHANGE# column in both query results. If the value of the UNRECOVERABLE_CHANGE# column in the primary database is greater than the same column in the standby database, then the datafile needs to be copied from the primary site to the standby site.

   In this example, the value of the UNRECOVERABLE_CHANGE# in the primary database for the `tbs_1.dbf` datafile is greater, so you need to copy the `tbs_1.dbf` datafile to the standby site.

**Step 2 On the primary site, back up the datafile that you need to copy to the standby site as follows.**

Issue the following SQL statements:

```
SQL> ALTER TABLESPACE system BEGIN BACKUP;
SQL> EXIT;
% cp tbs_1.dbf /backup
SQL> ALTER TABLESPACE system END BACKUP;
```

**Step 3 On the standby database, restart managed recovery.**

Issue the following SQL statement:

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

You might get the following error messages (possibly in the alert log) when you try to restart managed recovery:

```
ORA-00308: cannot open archived log 'standby1'
ORA-27037: unable to obtain file status
SVR4 Error: 2: No such file or directory
Additional information: 3
ORA-01547: warning: RECOVER succeeded but OPEN RESETLOGS would get error below
ORA-01152: file 1 was not restored from a sufficiently old backup
ORA-01110: data file 1: '/oracle/dbs/stdby/tbs_1.dbf'
```

If you get the ORA-00308 error, cancel recovery by issuing the following statement from another terminal window:

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

These error messages are returned when one or more logs in the archive gap have not been successfully applied. If you receive these errors, manually resolve the gaps and repeat step 3.

> **See Also:** Section B.3 for information on manually resolving an archive gap

### 10.5.3 Determining If a Backup Is Required After Unrecoverable Operations

If you performed unrecoverable operations on your primary database, determine if a new backup operation is required by following these steps:

1. Query the V$DATAFILE view on the primary database to determine the **system change number (SCN)** or the time at which the Oracle database server generated the most recent invalided redo data.
2. Issue the following SQL statement on the primary database to determine if you need to perform another backup:

   ```
   SELECT UNRECOVERABLE_CHANGE#,
          TO_CHAR(UNRECOVERABLE_TIME, 'mm-dd-yyyy hh:mi:ss')
   FROM   V$DATAFILE;
   ```

3. If the query in the previous step reports an unrecoverable time for a datafile that is more recent than the time when the datafile was last backed up, then make another backup of the datafile in question.

> **See Also:** V$DATAFILE in Chapter 14 and the *Oracle9i Database Reference* for more information about the V$DATAFILE view

# Part II

## Reference

This part provides reference material to be used in conjunction with the Oracle Data Guard standby database features. For more complete reference material, refer to the Oracle9*i* documentation set.

This part contains the following chapters:

# 11

# Initialization Parameters

This chapter describes how to view the current database initialization parameters, how to modify a server parameter file (SPFILE), and provides reference information for the initialization parameters that affect instances in a Data Guard configuration.

All database initialization parameters are contained in either an initialization parameter file (PFILE) or a server parameter file (SPFILE). As an alternative to specifying parameters in an initialization parameter file or server parameter file, you can modify dynamic parameters at runtime using the `ALTER SYSTEM SET` or `ALTER SESSION SET` statements.

---

**Note:** You must use a server parameter file if you use the Data Guard broker. Also, any runtime parameter modifications that are not recorded in either the initialization parameter file or the server parameter file are not persistent, and will be lost the next time the database is restarted.

---

## 11.1 Viewing Initialization Parameters

The following table describes the methods you can use to view the current initialization parameter settings:

| Method | Description |
|---|---|
| SHOW PARAMETERS SQL*Plus command | Issue this command to display the parameter values that are currently in effect. |
| V$PARAMETER view | Query this view to display the parameter values that are currently in effect. |
| V$PARAMETER2 view | Query this view to display the parameter values that are currently in effect. The output from this view is the same, but more readable, than the output from the V$PARAMETER view. |
| V$SPPARAMETER view | Query this view to display the current contents of the server parameter file. The view returns null values if a server parameter file is not being used by the instance. |

The following example queries the V$PARAMETER view for the CONTROL_FILES parameter setting:

```
SQL> SELECT NAME, VALUE FROM V$PARAMETER WHERE NAME = 'CONTROL_FILES';
```

## 11.2 Modifying a Server Parameter File

The server parameter file is a binary file and therefore cannot be edited manually. To change values in the server parameter file, you must export it to an editable format, make changes, and then import it back into a server parameter file, or use the ALTER SYSTEM SET statement to change the server parameter values. These methods are described in the following sections.

### 11.2.1 Exporting a Server Parameter File to an Editable File for Modifications

To modify a server parameter file do the following:

1. Use the SQL CREATE PFILE statement to export the server parameter file to a text initialization parameter file, as shown in Example 1 and Example 2 (that follow this list).

   An initialization parameter file is a text file and can therefore be edited manually. You must have the SYSDBA or the SYSOPER system privilege to execute this statement. The exported file is created on the database server

system. It contains any comments associated with the parameter in the same line as the parameter setting.

2. Edit the initialization parameter file.
3. Use the SQL CREATE SPFILE statement to create a new server parameter file from the edited initialization parameter file, as shown in Example 3 and Example 4 (that follow this list).

   You must have the SYSDBA or the SYSOPER system privilege to execute this statement.

### Example 1

This example creates a text initialization parameter file from the server parameter file without specifying filenames:

```
CREATE PFILE FROM SPFILE;
```

Because no names are specified for the files, an operating system-specific name is used for the initialization parameter file, and it is created from the operating system-specific default server parameter file.

### Example 2

This example creates a text initialization parameter file from a server parameter file where the names of the files are specified:

```
SQL> CREATE PFILE='/u01/oracle/dbs/test_init.ora'
  2> FROM SPFILE='/u01/oracle/dbs/test_spfile.ora';
```

### Example 3

This example creates a server parameter file from the initialization parameter file `/u01/oracle/dbs/test_init.ora`. An SPFILE name is not specified, so the file is created using an operating system-specific default server parameter filename and location:

```
SQL> CREATE SPFILE FROM PFILE='/u01/oracle/dbs/test_init.ora';
```

### Example 4

This example creates a server parameter file and supplies a name for both the server parameter file and the initialization parameter file:

```
SQL> CREATE SPFILE='/u01/oracle/dbs/test_spfile.ora'
  2> FROM PFILE='/u01/oracle/dbs/test_init.ora';
```

### 11.2.2 Using SQL ALTER SYSTEM SET to Modify a Server Parameter File

As an alternative to exporting, editing, and importing the server parameter file, as described in the previous section, you can use the SQL ALTER SYSTEM SET statement to change initialization parameter values. Make sure that you use the SCOPE clause to apply the change in the server parameter file.

By default, the scope is set to BOTH if a server parameter file was used to start up the instance, and the scope is set to MEMORY if an initialization parameter file was used to start up the instance. The following example adds a new local archive log destination to the server parameter file:

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_4=
  2> 'LOCATION=/disk1/oracle/oradata/payroll/',
  3> 'MANDATORY', 'REOPEN=2'  SCOPE=SPFILE;
```

## 11.3 Initialization Parameters for Instances in a Data Guard Configuration

The following list shows the initialization parameters that affect instances in a Data Guard environment:

- ARCHIVE_LAG_TARGET
- COMPATIBLE
- CONTROL_FILE_RECORD_KEEP_TIME
- CONTROL_FILES
- DB_FILE_NAME_CONVERT
- DB_FILES
- DB_NAME
- FAL_CLIENT
- FAL_SERVER
- LOCK_NAME_SPACE
- LOG_ARCHIVE_DEST_n
- LOG_ARCHIVE_DEST_STATE_n
- LOG_ARCHIVE_FORMAT
- LOG_ARCHIVE_MAX_PROCESSES

- LOG_ARCHIVE_MIN_SUCCEED_DEST
- LOG_ARCHIVE_START
- LOG_ARCHIVE_TRACE
- LOG_FILE_NAME_CONVERT
- LOG_PARALLELISM
- PARALLEL_MAX_SERVERS
- REMOTE_ARCHIVE_ENABLE
- SHARED_POOL_SIZE
- SORT_AREA_SIZE
- STANDBY_ARCHIVE_DEST
- STANDBY_FILE_MANAGEMENT
- USER_DUMP_DEST

The following sections provide a description for each parameter that indicates if the parameter applies to the primary database role, the standby database role, or both. For parameters that apply to the standby database role, most of the parameters pertain to both physical and logical standby databases. Any differences are noted.

> **See Also:** *Oracle9i Database Reference* for information about these parameters that is not specific to Data Guard and for the type, default values, and syntax for these initialization parameters. Also refer to your Oracle operating system-specific documentation for more information about setting initialization parameters.

# ARCHIVE_LAG_TARGET

## Description

Limits the amount of data that can be lost and effectively increases the availability of the standby database by forcing a log switch after the amount of time you specify (in seconds) elapses. The standby database will not miss redo logs generated from a time range longer than a value of the ARCHIVE_LAG_TARGET parameter.

## Role

Applies to the primary database role

## Examples

The following example sets the log switch interval to 30 minutes (a typical value):

```
ARCHIVE_LAG_TARGET = 1800
```

# COMPATIBLE

## Description

Controls the compatibility of your database. Set to 9.0.0.0.0 or higher to use the Data Guard broker, logical standby databases, and the enhanced features of physical standby databases. Always set this parameter to the same value on the primary database and standby databases. If the values differ, you might not be able to archive the redo logs from the primary database to the standby database.

## Role

Applies to the primary and standby database roles

## Examples

The following example sets the database compatible level to '9.2.0.0.0':

```
COMPATIBLE = '9.2.0.0.0'
```

# CONTROL_FILE_RECORD_KEEP_TIME

## Description

Specifies the minimum number of days before a reusable record in the control file can be reused. Use this parameter to avoid overwriting a reusable record in the control file (that contains needed information such as an archive log) for a specified period of time. The range of values for this parameter is 0 to 356 days. If this parameter is set to 0, then the reusable records are reused as needed.

## Role

Applies to the primary and standby database roles

## Examples

The following example sets the minimum number of days before a reusable record in the control file can be reused to 20 days:

```
CONTROL_FILE_RECORD_KEEP_TIME = 20
```

# CONTROL_FILES

## Description

Specifies the names of one or more control files, separated by commas. Always set this parameter on the standby database to a different value than the CONTROL_FILES parameter for the primary database, if these databases are on the same system. The filenames you specify with the CONTROL_FILES parameter for the standby database must exist at the standby location.

## Role

Applies to the primary and standby database roles

## Examples

The following example specifies two control files for the database instance:

```
CONTROL_FILE = ("/disk1/oracle/oradata/payroll/control01.ctl",
"/disk1/oracle/oradata/payroll/control02.ctl")
```

# DB_FILE_NAME_CONVERT

## Description

Converts the filename of a datafile on the primary database to a filename on the standby database. Because the standby database control file is a copy of the primary database control file, you must use this parameter to convert the standby database filenames when they are different from the primary database filenames. If the standby database is on the same system as the primary database, you must use different path names.

## Role

Applies to the physical standby database role

## Examples

The following example shows the conversion of paths from `/dbs/t1/` (primary database) to `/dbs/t1/stdby` (standby database) and `dbs/t2/` (primary database) to `dbs/t2/stdby` (standby database):

```
DB_FILE_NAME_CONVERT = ('/dbs/t1/','/dbs/t1/stdby','dbs/t2/ ','dbs/t2/stdby')
```

# DB_FILES

## Description

Specifies the maximum number of database files that can be open for this database. The primary and standby databases should have the same value for this parameter.

## Role

Applies to the primary and standby database roles

## Examples

The following example specifies that a maximum of 300 database files can be open for this database instance:

```
DB_FILES = 300
```

# DB_NAME

## Description

Specifies a database identifier of up to eight characters. For a physical standby database, set the DB_NAME parameter to the same value as it is set in the primary database initialization file. For a logical standby database, set the DB_NAME parameter to a different value from that in the primary database initialization files. Use the DBNEWID (nid) utility to set the database name for a logical standby database, as described in Section 4.2.14.

## Role

Applies to the primary and standby database roles

## Examples

The following example shows that the database name is Sales:

```
DB_NAME = Sales
```

The following example shows how to use the DBNEWID utility to set a logical standby database name. You must mount the database before issuing this command.

```
nid TARGET=SYS/CHANGE_ON_INSTALL@LogicalSDB DBNAME=SalesLSDB SETNAME=YES
```

# FAL_CLIENT

## Description

Assigns the fetch archive log (FAL) client name used by the FAL server to refer to the FAL client. This is the Oracle Net service name that the FAL server should use to refer to the standby database. This Oracle Net service name must be configured properly on the FAL server (primary database) to point to the FAL client. Given the dependency of the FAL_CLIENT parameter on the FAL_SERVER parameter, the two parameters should be configured or changed at the same time. This parameter is set on the standby site.

## Role

Applies to the physical standby database role in managed recovery mode

## Examples

The following example assigns the FAL client to the Oracle Net service name StandbyDB:

```
FAL_CLIENT = StandbyDB
```

# FAL_SERVER

## Description

Assigns the Oracle Net service name that the standby database should use to connect to the fetch archive log (FAL) server. This parameter is set on the standby system.

## Role

Applies to the physical standby database role in managed recovery mode

## Examples

The following example shows that the FAL server is assigned to the Oracle Net service name `PrimaryDB`:

```
FAL_SERVER = PrimaryDB
```

# LOCK_NAME_SPACE

## Description

Specifies the name space that the distributed lock manager (DLM) uses to generate lock names. Set this parameter to a unique value in each initialization parameter file if the standby database has the same name as the primary database and is on the same system or cluster.

> **Note:** If you do not set the LOCK_NAME_SPACE parameter differently when the standby and primary databases are located on the same system, you will receive an ORA-1102 error.

## Role

Applies to the primary and standby database roles

## Examples

The following example shows that the LOCK_NAME_SPACE is set to payroll2 in the standby initialization parameter file:

```
LOCK_NAME_SPACE = payroll2
```

# LOG_ARCHIVE_DEST_*n*

## Description

Defines an archive log destination and attributes for log transport services. This parameter is discussed in Chapter 5 and in Chapter 12.

## Role

Applies to the primary and standby database roles

## Examples

The following example shows a remote archive log destination to a standby database:

```
LOG_ARCHIVE_DEST_2 = 'SERVICE=payroll2 OPTIONAL REOPEN=180'
```

# LOG_ARCHIVE_DEST_STATE_*n*

## Description

Specifies the state of the destination specified by the LOG_ARCHIVE_DEST_*n* parameter. The possible values are as follows:

- ENABLE specifies that a valid log archive destination can be used for a subsequent archiving operation (automatic or manual). This is the default.
- DEFER specifies that valid destination information and attributes are preserved, but the destination is excluded from archiving operations until you reenable archiving with the ENABLE option.
- ALTERNATE specifies that the destination is not enabled, but will become enabled if communication to another destination fails.

## Role

Applies to the primary and standby database roles

## Examples

The following example shows the LOG_ARCHIVE_DEST_STATE_2 state is set to ENABLE:

```
LOG_ARCHIVE_DEST_STATE_2 = ENABLE
```

# LOG_ARCHIVE_FORMAT

## Description

Specifies the format for archived redo log filenames. STANDBY_ARCHIVE_DEST and LOG_ARCHIVE_FORMAT initialization parameters are concatenated to generate fully-qualified standby database archived redo log filenames.

## Role

Applies to the primary and standby database roles

## Examples

The following example specifies the format for the archive redo log filename using a database ID (%d), thread (%t), and sequence number (%s):

```
LOG_ARCHIVE_FORMAT = 'log%d_%t_%s.arc'
```

# LOG_ARCHIVE_MAX_PROCESSES

## Description

Specifies the number of archiver background processes to be invoked by the database server. This value is evaluated at instance startup if the LOG_ARCHIVE_START parameter has the value TRUE; otherwise, this parameter is evaluated when the archiver process is invoked.

## Role

Applies to the primary and standby database roles

## Examples

The following example sets the number of log archiver processes to 2:

```
LOG_ARCHIVE_MAX_PROCESSES = 2
```

# LOG_ARCHIVE_MIN_SUCCEED_DEST

## Description

Defines the minimum number of destinations that must receive redo logs successfully before the log writer process on the primary database can reuse the online redo logs.

## Role

Applies to the primary and standby database roles

## Examples

The following example sets the minimum number of destinations that must succeed to 2:

```
LOG_ARCHIVE_MIN_SUCCEED_DEST = 2
```

# LOG_ARCHIVE_START

## Description

Indicates if archiving should be automatic or manual when the instance starts up. To enable automatic archiving of filled log groups, set LOG_ARCHIVE_START in the initialization parameter file to TRUE. To disable the automatic archiving of filled online redo log groups, set LOG_ARCHIVE_START to FALSE. You cannot specify this parameter in a server parameter file.

## Role

Applies to the primary and standby database roles

## Examples

The following example sets LOG_ARCHIVE_START to TRUE:

```
LOG_ARCHIVE_START = TRUE
```

# LOG_ARCHIVE_TRACE

## Description

Controls trace output generated by the ARC*n* and LGWR processes and foreground processes on the primary database, and the RFS and FAL server processes on the standby database. It allows you to see the progression of the archivedd redo logs to the standby site. The Oracle database server writes an audit trail of the redo logs received from the primary database into a trace file. You specify the location of the trace file using the USER_DUMP_DEST parameter. Possible values include:

| Level | Meaning |
|---|---|
| 0 | Disables archived redo log tracing (default setting) |
| 1 | Tracks archiving of redo log file |
| 2 | Tracks archival status per archived redo log destination |
| 4 | Tracks archival operational phase |
| 8 | Tracks archived redo log destination activity |
| 16 | Tracks detailed archived redo log destination activity |
| 32 | Tracks archived redo log destination parameter modifications |
| 64 | Tracks ARC*n* process state activity |
| 128 | Tracks FAL server process activity |
| 256 | Supported in a future release |
| 512 | Tracks asynchronous LGWR activity |
| 1024 | Tracks the RFS physical client |
| 2048 | Tracks the ARC*n* or RFS heartbeat |

## Role

Applies to the primary and standby database roles

## Example

The following example sets the LOG_ARCHIVE_TRACE to 1:

```
LOG_ARCHIVE_TRACE = 1
```

# LOG_FILE_NAME_CONVERT

## Description

Converts the filename of a log on the primary database to the filename of a log on the standby database. Adding a log to the primary database necessitates adding a corresponding log to the standby database. When the standby database is updated, this parameter is used to convert the log filename from the primary database to the log filename on the standby database. This parameter is necessary when the standby database uses different path names from the primary database. If the standby database is on the same system as the primary database, you must use different path names.

## Role

Applies to the physical standby database roles

## Examples

The following example shows the conversion of two paths. It converts `/dbs/t1/` (primary database) to `/dbs/t1/stdby` (standby database) and `dbs/t2/` (primary database) to `dbs/t2/stdby` (standby database):

```
LOG_FILE_NAME_CONVERT = ('/dbs/t1/','/dbs/t1/stdby','dbs/t2/ ','dbs/t2/stdby')
```

# LOG_PARALLELISM

## Description

Specifies the level of concurrency for redo data allocation to allow parallel generation of redo data. Set this value to 1 for the primary database and for all logical standby databases, The default value is 1.

## Role

Applies to the logical standby database role only

## Examples

The following example sets the LOG_PARALLELISM parameter to 1:

```
LOG_PARALLELISM = 1
```

# PARALLEL_MAX_SERVERS

## Description

This parameter specifies the maximum number of parallel servers that can work on log apply services on the logical standby database. This parameter is not used with physical standby databases.

Log apply services use parallel query processes to perform processing, and use parallel apply algorithms to maintain a high level of database apply performance. A minimum of 5 parallel query processes is required for a logical standby database. Thus, the value of the PARALLEL_MAX_SERVERS parameter must be set to a value of 5 or greater.

## Role

Applies to the primary and logical standby database roles

## Examples

The following example sets the PARALLEL_MAX_SERVERS initialization parameter to 10:

```
PARALLEL_MAX_SERVERS = 10
```

# REMOTE_ARCHIVE_ENABLE

## Description

Enables or disables the sending of redo logs to remote destinations and the receipt of remote redo logs.

Possible values are:

- TRUE

  Enables the sending of redo logs to a remote destination or receipt of remote redo logs. Set this parameter to TRUE on the primary database and standby databases in the Data Guard environment to allow the primary database to send redo logs to the standby database and to allow the standby database to receive redo logs for archiving from the primary database.

- FALSE

  Disables both the sending and receiving of redo logs.

- SEND

  Enables the primary database to send redo logs to the standby database.

- RECEIVE

  Enables the standby database to receive redo logs from the primary database.

To independently enable and disable the sending and receiving of remote redo logs, use the send and receive values. The SEND and RECEIVE values together are the same as specifying true. Every instance of an Oracle Real Application Clusters database must contain the same REMOTE_ARCHIVE_ENABLE value.

## Role

Applies to the primary and standby database roles

## Examples

The following example enables the remote log sending or receiving:

```
REMOTE_ARCHIVE_ENABLE = true
```

# SHARED_POOL_SIZE

## Description

Specifies (in bytes) the size of the shared pool. Log apply services of logical standby databases use a shared pool system global area (SGA) to stage the information read from the redo logs. The more SGA that is available, the more information that can be staged. By default, one quarter of the value set for the SHARED_POOL_SIZE parameter will be used by log apply services. You can change this default using the DBMS_LOGSTDBY.APPLY_SET PL/SQL procedure.

## Role

Applies to the primary and standby database roles

## Examples

The following example sets the shared pool size to 33 MB:

```
SHARED_POOL_SIZE = 33554432
```

# SORT_AREA_SIZE

## Description

Specifies in bytes the maximum amount of memory the Oracle database server will use for a sort operation. Set this parameter to a value that allows you to execute the `SELECT * FROM V$PARAMETER` statement when the database is not open. This prevents errors if you attempt to sort without temporary tablespaces when the database is not open.

## Role

Applies to the primary and standby database roles

## Examples

The following example sets the sort area size to 65536 bytes:

```
SORT_AREA_SIZE = 65536
```

# STANDBY_ARCHIVE_DEST

## Description

Used by a standby database to determine the archive location of online redo logs received from the primary database. The RFS process uses this value in conjunction with the LOG_ARCHIVE_FORMAT value to generate the fully-qualified standby database redo log filenames. Note that the generated filename is overridden by the TEMPLATE attribute of the LOG_ARCHIVE_DEST_*n* parameter.

You can see the value of this parameter by querying the V$ARCHIVE_DEST data dictionary view.

## Role

Applies to the standby database role

## Examples

The following example specifies that '/u01/oracle/oradata/archive' is the redo log file path on the standby database:

```
STANDBY_ARCHIVE_DEST = '/u01/oracle/oradata/archive'
```

# STANDBY_FILE_MANAGEMENT

## Description

Enables or disables automatic standby file management.

The possible values for this parameter are:

- MANUAL

  Disables automatic standby file management

- AUTO

  Enables automatic standby file management

When set to AUTO, this parameter automates the creation and deletion of datafile filenames on the standby site using the same filenames as the primary site. When set to MANUAL, datafile creation and deletion are not automated and might cause managed recovery operations to terminate.

Use this parameter with the DB_FILE_NAME_CONVERT initialization parameter to ensure that the correct files are created on the standby site when the standby database has a different file path from the primary database. Note that this parameter does not support datafile filenames on RAW devices.

## Role

Applies to the primary and standby database roles

## Examples

The following example enables automatic standby file management:

```
STANDBY_FILE_MANAGEMENT = TRUE
```

# USER_DUMP_DEST

## Description

Specifies the directory path name where the database server will write debugging trace files on behalf of a user process. Use the LOG_ARCHIVE_TRACE parameter to control the trace information.

## Role

Applies to the primary and standby database roles

## Examples

The following example specifies the location for the database trace files to be '/u01/oracle/oradata/utrc':

```
USER_DUMP_DEST = '/u01/oracle/oradata/utrc'
```

# 12

# LOG_ARCHIVE_DEST_*n* Parameter Attributes

This chapter provides syntax, values, and information on validity for the archival attributes of the LOG_ARCHIVE_DEST_*n* initialization parameter. These attributes include:

AFFIRM and NOAFFIRM
ALTERNATE and NOALTERNATE
ARCH and LGWR
DELAY and NODELAY
DEPENDENCY and NODEPENDENCY
LOCATION and SERVICE
MANDATORY and OPTIONAL
MAX_FAILURE and NOMAX_FAILURE
NET_TIMEOUT and NONET_TIMEOUT
QUOTA_SIZE and NOQUOTA_SIZE
QUOTA_USED and NOQUOTA_USED
REGISTER and NOREGISTER
REGISTER=location_format
REOPEN and NOREOPEN
SYNC and ASYNC
TEMPLATE and NOTEMPLATE

In addition, this chapter includes the following topics:

- About LOG_ARCHIVE_DEST_n Parameter Attributes
- Attribute Compatibility for Archive Destinations

**Note:** Each destination *must* identify either a local disk directory or a remotely accessed database. See Chapter 5 for additional information about log transport services and using these initialization parameters. See Appendix D for information about using these initialization parameters to set up cascaded redo logs.

## 12.1 About LOG_ARCHIVE_DEST_*n* Parameter Attributes

You should specify at least two LOG_ARCHIVE_DEST_*n* (where *n* is an integer from 1 to 10) parameters: one for the required local destination and another for a local or remote destination.

The following sections describe the attributes of the LOG_ARCHIVE_DEST_*n* initialization parameter. See Chapter 5 for additional information.

All LOG_ARCHIVE_DEST_*n* parameters must contain, at a minimum, either a LOCATION or SERVICE attribute. In addition, you must have a LOG_ARCHIVE_DEST_STATE_*n* parameter for each defined destination.

The LOG_ARCHIVE_DEST_STATE_*n* (where *n* is an integer from 1 to 10) initialization parameter specifies the state of the corresponding destination indicated by the LOG_ARCHIVE_DEST_*n* initialization parameter. For example, the LOG_ARCHIVE_DEST_STATE_3 parameter specifies the state of the LOG_ARCHIVE_DEST_3 destination.

## 12.2 Changing Destination Attributes Using SQL Statements

Table 12–1 lists the attributes that can be set for the LOG_ARCHIVE_DEST_*n* initialization parameter and indicates if the attribute can be changed using an ALTER SYSTEM or ALTER SESSION statement.

***Table 12–1 Changing Destination Attributes Using SQL***

| Attribute | ALTER SYSTEM | ALTER SESSION |
|---|---|---|
| [NO]AFFIRM | Yes | Yes |
| [NO]ALTERNATE=*destination* | Yes | Yes |
| ARCH | Yes | Yes |
| ASYNC[=*blocks*] | Yes | No |
| [NO]DELAY | Yes | Yes |

***Table 12–1 Changing Destination Attributes Using SQL***

| Attribute | ALTER SYSTEM | ALTER SESSION |
|---|---|---|
| [NO]DEPENDENCY=*destination* | Yes | No |
| LGWR | Yes | No |
| LOCATION=*local_disk_directory* | Yes | Yes |
| MANDATORY | Yes | Yes |
| [NO]MAX_FAILURE=*count* | Yes | No |
| OPTIONAL | Yes | Yes |
| [NO]NET_TIMEOUT[=*seconds*] | Yes | No |
| [NO]QUOTA_SIZE=*blocks* | Yes | No |
| [NO]QUOTA_USED=*blocks* | Yes | No |
| [NO]REGISTER | Yes | Yes |
| REGISTER=location_format | Yes | Yes |
| [NO]REOPEN[=*seconds*] | Yes | Yes |
| SERVICE=*net_service_name* | Yes | Yes |
| SYNC[=PARALLEL\|NOPARALLEL] | Yes | Yes |
| [NO]TEMPLATE=*filename_template* | Yes | Yes |

## 12.3 Incrementally Changing LOG_ARCHIVE_DEST_*n* Parameter Settings

The log transport services LOG_ARCHIVE_DEST_*n* destination initialization parameters are unique in that they contain multiple values known as **attributes**. Except for the LOCATION and SERVICE attributes, all attributes are optional and have a default value.

---

**Note:** When using a traditional text initialization parameter file, the LOG_ARCHIVE_DEST_*n* parameters can also be specified incrementally at runtime so that you can replace one or more specific attributes without having to re-specify the entire parameter value. See Section 5.8.1 for examples that show incremental changes to a traditional text initialization parameter file.

---

To specify network service names that use embedded characters, such as equal signs (=) and spaces, enclose the service name in quotation marks (").

Example 12–1 shows how to replace the initial specification of the LOG_ARCHIVE_DEST_1 parameter.

***Example 12–1 Replacing a Destination Specification***

```
LOG_ARCHIVE_DEST_1='LOCATION=/disk1/oracle/oradata/payroll'
LOG_ARCHIVE_DEST_2='SERVICE=stdby REOPEN=60'
LOG_ARCHIVE_DEST_1='LOCATION=/disk3/oracle/oradata/payroll MANDATORY'
```

When using a traditional text initialization parameter file, the LOG_ARCHIVE_DEST_*n* initialization parameters can be changed incrementally at run-time using the ALTER SYSTEM SET statement. This means that you are able to change one or more specific attributes without having to re-specify the entire parameter value. Specifying any attribute except the LOCATION or SERVICE attributes is valid for an incremental change.

---

**Note:** You cannot make incremental changes to parameters in a server parameter file (SPFILE). You must convert the SPFILE to a traditional text initialization parameter file (PFILE) and edit the initialization parameters, then convert the PFILE back to a SPFILE.

---

Example 12–2 shows how to set multiple attributes incrementally on separate lines. Specify the SERVICE or LOCATION attribute on the first line.

***Example 12–2 Specifying Multiple Attributes Incrementally***

```
ALTER SYSTEM SET LOG_ARCHIVE_DEST_1='LOCATION=/disk1/oracle/oradata/payroll';
ALTER SYSTEM SET LOG_ARCHIVE_DEST_1='OPTIONAL';
ALTER SYSTEM SET LOG_ARCHIVE_DEST_1='REOPEN=5';
```

Example 12–3 shows how to specify attributes for multiple destinations. Incremental parameters such as the LOG_ARCHIVE_DEST_*n* initialization parameter do not have to immediately follow each other.

***Example 12–3 Specifying Multiple Attributes for Multiple Destinations***

```
ALTER SYSTEM SET LOG_ARCHIVE_DEST_1='LOCATION=/disk1/oracle/oradata/payroll';
ALTER SYSTEM SET LOG_ARCHIVE_DEST_2='SERVICE=stdby REOPEN=60';
ALTER SYSTEM SET LOG_ARCHIVE_DEST_1= OPTIONAL';
```

Specifying the LOCATION or SERVICE attribute causes the destination initialization parameter to be reset to its default values. Example 12–4 shows an entry for LOG_ARCHIVE_DEST_1 that is not considered an incremental change:

***Example 12–4 Replaced Destination Specification***

```
ALTER SYSTEM SET LOG_ARCHIVE_DEST_1='LOCATION=/disk1/oracle/oradata/payroll
REOPEN=60';
ALTER SYSTEM SET LOG_ARCHIVE_DEST_1='LOCATION=/disk1/oracle/oradata/payroll';
```

A string containing a null value for parameter attributes clears a previously entered destination specification. Example 12–5 shows how to clear the definition of LOG_ARCHIVE_DEST_1.

***Example 12–5 Clearing a Destination Specification***

```
LOG_ARCHIVE_DEST_1='LOCATION=/disk1/oracle/oradata/payroll'
LOG_ARCHIVE_DEST_2='SERVICE=stdby REOPEN=60'
LOG_ARCHIVE_DEST_1=''
```

### 12.3.1 Viewing Current Settings of Destination Initialization Parameters

You can use SQL to query fixed views such as V$ARCHIVE_DEST to see current LOG_ARCHIVE_DEST_*n* initialization parameter settings. For example, to view current destination settings on the primary database, enter the following statement:

```
SQL> SELECT DESTINATION FROM V$ARCHIVE_DEST;
```

**Note:** Do not use the V$PARAMETER view to determine the value of the LOG_ARCHIVE_DEST_*n* initialization parameters. The V$PARAMETER view shows only the last specified value for each parameter, which in the case of an incremental modification is not representative of the actual LOG_ARCHIVE_DEST_*n* parameter value.

# AFFIRM and NOAFFIRM

| Category | AFFIRM | NOAFFIRM |
|---|---|---|
| Datatype of the attribute | Keyword | Keyword |
| Minimum value | Not applicable | Not applicable |
| Maximum value | Not applicable | Not applicable |
| Default value | Not applicable | Not applicable |
| Requires attributes ... | Not applicable | Not applicable |
| Conflicts with attributes ... | NOAFFIRM | AFFIRM |
| Attribute class | ALTER SESSION and ALTER SYSTEM | ALTER SESSION and ALTER SYSTEM |
| Corresponding V$ARCHIVE_DEST column | AFFIRM | AFFIRM |
| Related V$ARCHIVE_DEST column | ASYNC_BLOCKS | ASYNC_BLOCKS |

## Purpose

Use the AFFIRM and the NOAFFIRM attributes to ensure that archived redo log contents were successfully written to disk. This applies to both local and remote destinations.

## Defaults

If the AFFIRM or the NOAFFIRM attribute is not specified with the LOG_ARCHIVE_DEST_*n* parameter, the default is NOAFFIRM.

## Attributes

### AFFIRM

The AFFIRM attribute indicates that all archived redo log I/O operations are to be performed synchronously, even on a remote standby database. This attribute has the potential to affect primary database performance. When you use the LGWR and AFFIRM attributes to indicate that the log writer process synchronously writes the locally archived redo log contents to disk, control is not returned to the users and does not continue until the disk I/O operation has completed. When you use the

ARCH and AFFIRM attributes to indicate that the ARC*n* process will synchronously write the archived redo logs to disk, the archival operation might take longer, and online redo logs might not be reusable until archiving is complete.

Using the AFFIRM attribute does not affect performance when you use the ASYNC attribute.

Query the AFFIRM column of the V$ARCHIVE_DEST fixed view to see whether or not the AFFIRM attribute is being used for the associated destination.

The following table identifies the various combinations of these attributes and their potential for affecting primary database performance and data availability. For example, the AFFIRM attribute used in conjunction with the SYNC and ASYNC attributes provides the highest degree of data protection but results in negative performance on the primary database.

| Network I/O Attribute | Archived Redo Log Disk I/O Attribute | Potential Primary Database Performance | Standby Database Data Protection |
|---|---|---|---|
| SYNC | AFFIRM | Lowest | Highest |
| SYNC | NOAFFIRM | Low | High |
| ASYNC | AFFIRM | High | Low |
| ASYNC | NOAFFIRM | Highest | Lowest |

The highest degree of data availability also has the potential for the lowest primary database performance.

---

**Note:** When the primary database is in the MAXIMIZE PROTECTION or MAXIMIZE AVAILABILITY mode, redo log archiving destinations using the log writer process are also automatically placed in AFFIRM mode.

---

**See Also:** SYNC and ASYNC on page 12-48

### NOAFFIRM

The NOAFFIRM attribute indicates that all archived redo log disk I/O operations are to be performed asynchronously; the log writer process does not wait until the disk I/O has completed before continuing.

## Examples

The following example shows the AFFIRM attribute with the LOG_ARCHIVE_DEST_*n* parameter.

```
LOG_ARCHIVE_DEST_3='SERVICE=stby1 LGWR SYNC AFFIRM'
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

# ALTERNATE and NOALTERNATE

| Category | ALTERNATE=*destination* | NOALTERNATE |
|---|---|---|
| Datatype of the attribute | String value | Keyword |
| Minimum value | Not applicable | Not applicable |
| Maximum value | Not applicable | Not applicable |
| Default value | None[1] | Not applicable |
| Requires attributes ... | Not applicable | Not applicable |
| Conflicts with attributes ... | NOALTERNATE | ALTERNATE |
| Attribute class | ALTER SYSTEM | ALTER SESSION and ALTER SYSTEM |
| Corresponding V$ARCHIVE_DEST column | ALTERNATE | ALTERNATE |
| Related V$ARCHIVE_DEST column | STATUS | STATUS |

[1] If the NOALTERNATE attribute is specified, or if no alternate destination is specified, the destination does not automatically change to another destination upon failure.

## Purpose

The ALTERNATE and the NOALTERNATE attributes of the LOG_ARCHIVE_DEST_*n* parameter define an alternate archiving destination or prevent archiving to an alternate destination when the original archiving destination fails.

## Defaults

If the ALTERNATE or the NOALTERNATE attribute is not specified with the LOG_ARCHIVE_DEST_*n* parameter, the default is NOALTERNATE.

## Attributes

### ALTERNATE=*destination*

Use the ALTERNATE attribute of the LOG_ARCHIVE_DEST_*n* parameter to define an alternate archiving destination to be used if archiving to the original archiving destination fails.

An archiving destination can have a maximum of one alternate destination specified. An alternate destination is used when the transmission of an online redo log from the primary site to the standby site fails. If archiving fails and the REOPEN attribute is specified with a value of zero (0), or NOREOPEN is specified, the Oracle database server attempts to archive online redo logs to the alternate destination on the next archival operation.

An alternate destination can reference a local or remote archiving destination. An alternate destination cannot be self-referencing.

A destination can also be in the ALTERNATE state; this state is specified using the LOG_ARCHIVE_DEST_STATE_*n* initialization parameter. The ALTERNATE state defers processing of the destination until such time as another destination failure automatically enables this destination, if the alternate destination attributes are valid. See Section 5.4 for more details about the LOG_ARCHIVE_DEST_STATE_*n* parameter.

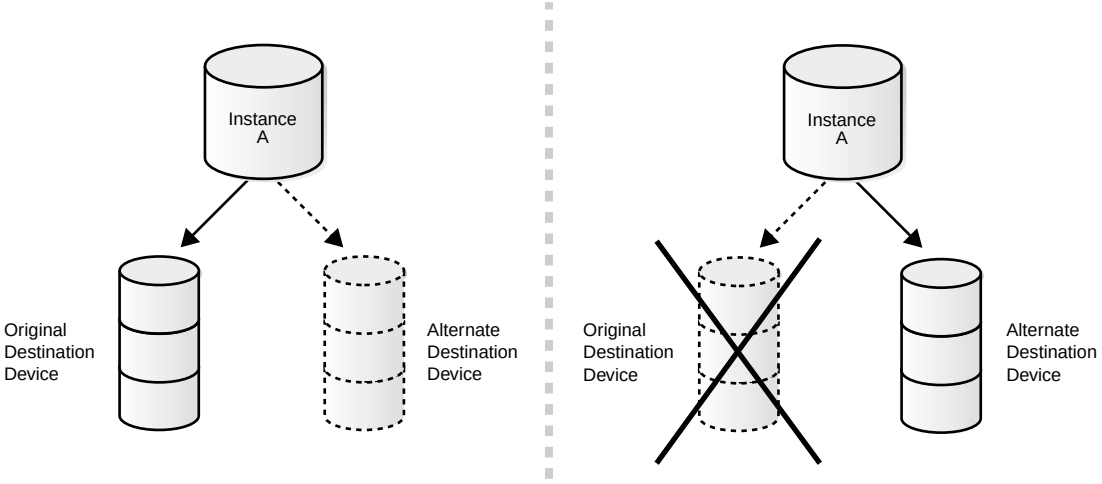
The ALTERNATE attribute cannot be modified at the session level.

In the following example, if the LOG_ARCHIVE_DEST_1 destination fails, the archiving process automatically switches to the LOG_ARCHIVE_DEST_2 destination.

```
LOG_ARCHIVE_DEST_1='LOCATION=/disk1 MANDATORY ALTERNATE=LOG_ARCHIVE_DEST_2'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_2='LOCATION=/disk2 MANDATORY'
LOG_ARCHIVE_DEST_STATE_2=ALTERNATE
```

Figure 12–1 shows a scenario where online redo logs are archived to a local disk device. If the original destination device becomes full or unavailable, the archival operation is automatically redirected to the alternate destination device.

***Figure 12–1 Archiving Operation to an Alternate Destination Device***




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

- The NOREOPEN attribute is specified.
- A value of zero (0) is specified for the REOPEN attribute.
- A nonzero REOPEN attribute and a nonzero MAX_FAILURE count were exceeded.

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

The following is the archived redo log destination attribute precedence table:

| Precedence[1] | Attribute |
|---|---|
| 1 | MAX_FAILURE |
| 2 | REOPEN |
| 3 | ALTERNATE |
| 4 | MANDATORY |

[1] 1 indicates highest; 4 indicates lowest.

The use of a standby database as the target of an alternate destination should be carefully handled. Ideally, a standby alternate destination should only be used to specify a different network route to the same standby database system.

If no enabled destination references the alternate destination, the alternate destination is implied to be deferred, because there is no automatic method of enabling the alternate destination.

An alternate destination can be manually enabled at runtime. Conversely, an alternate destination can be manually deferred at runtime. See Section 5.8.1.2 for more information about changing initialization parameter settings using SQL at runtime.

There is no general pool of alternate archived redo log destinations. Ideally, for any enabled destination, the database administrator should choose an alternate destination that closely mirrors that of the referencing destination, although that is not required.

Each enabled destination can have its own alternate destination. Conversely, several enabled destinations can share the same alternate destination. This is known as an overlapping set of destinations. Enabling the alternate destination determines the set to which the destination belongs.

Increasing the number of enabled destinations decreases the number of available alternate redo log archiving destinations.

---

**Note:** An alternate destination is enabled for the next archival operation. There is no support for enabling the alternate destination in the middle of the archival operation because that would require rereading already processed blocks, and so forth. This is identical to the `REOPEN` attribute behavior.

---

Any destination can be designated as an alternate given the following restrictions:

- At least one local mandatory destination is enabled.
- The number of enabled destinations must meet the defined `LOG_ARCHIVE_MIN_SUCCEED_DEST` parameter value.
- A destination cannot be its own alternate; however, this does not generate an error.

Destinations defined using the SQL ALTER SESSION statement do not activate an alternate destination defined at the system level. Conversely, system-defined destinations do not activate an alternate destination defined at the session level.

If the REOPEN attribute is specified with a nonzero value, the ALTERNATE attribute is ignored. If the MAX_FAILURE attribute is also specified with a nonzero value, and the failure count exceeds the specified failure threshold, the ALTERNATE destination is enabled. Therefore, the ALTERNATE attribute does not conflict with a nonzero REOPEN attribute value.

**NOALTERNATE**

Use the NOALTERNATE attribute of the LOG_ARCHIVE_DEST_*n* parameter to prevent the original destination from automatically changing to an alternate destination when the original destination fails.

## Examples

In the sample initialization parameter file in Example 12–6, LOG_ARCHIVE_DEST_1 automatically fails over to LOG_ARCHIVE_DEST_2 on the next archival operation if an error occurs or the device becomes full.

***Example 12–6 Automatically Failing Over to an Alternate Destination***

```
LOG_ARCHIVE_DEST_1=
'LOCATION=/disk1 MANDATORY NOREOPEN ALTERNATE=LOG_ARCHIVE_DEST_2'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_2='LOCATION=/disk2 MANDATORY'
LOG_ARCHIVE_DEST_STATE_2=ALTERNATE
```

The sample initialization parameter file in Example 12–7 shows how to define an alternate Oracle Net service name to the same standby database.

***Example 12–7 Defining an Alternate Oracle Net Service Name to the Same Standby Database***

```
LOG_ARCHIVE_DEST_1='LOCATION=/disk1 MANDATORY'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_2='SERVICE=stby1_path1 NOREOPEN OPTIONAL ALTERNATE=LOG_ARCHIVE_DEST_3'
LOG_ARCHIVE_DEST_STATE_2=ENABLE
LOG_ARCHIVE_DEST_3='SERVICE=stby1_path2 NOREOPEN OPTIONAL'
LOG_ARCHIVE_DEST_STATE_3=ALTERNATE
```

# ARCH and LGWR

| Category | ARCH | LGWR |
|---|---|---|
| Datatype of the attribute | Keyword | Keyword |
| Minimum value | Not applicable | Not applicable |
| Maximum value | Not applicable | Not applicable |
| Default value | Not applicable | Not applicable |
| Requires attributes ... | Not applicable | Not applicable |
| Conflicts with attributes ... | LGWR, ASYNC, NET_TIMEOUT | ARCH |
| Attribute class | ALTER SESSION and ALTER SYSTEM | ALTER SYSTEM only |
| Corresponding V$ARCHIVE_DEST column | ARCHIVER | ARCHIVER |
| Related V$ARCHIVE_DEST columns | PROCESS, SCHEDULE | PROCESS, SCHEDULE |

## Purpose

The optional ARCH and LGWR attributes are used to specify that either the archiver or log writer process is responsible for transmitting online redo logs to local and remote archival destinations.

When you change the archiving process from the ARC*n* process to the LGWR process using the ARCH and LGWR attributes for an archive destination, the LGWR process does not start archiving until the next log switch operation. Conversely, when you change the archiving process from the LGWR process to the ARC*n* process, the LGWR process continues to archive until the next log switch operation.

## Defaults

If the ARCH or the LGWR attribute is not specified with the LOG_ARCHIVE_DEST_*n* parameter, the default is ARCH.

## Attributes

### ARCH

The ARCH attribute indicates that redo logs are transmitted to the destination during an archival operation. The background archiver processes (ARC*n*) or a foreground archival operation serves as the redo log transport service.

### LGWR

The LGWR attribute indicates that redo logs are transmitted to the destination concurrently as the online redo log is populated. The background log writer process (LGWR) serves as the redo log transport service. When transmitting redo logs to remote destinations, the LGWR process establishes a network connection to the destination instance. Because the redo logs are transmitted concurrently, they are not retransmitted to the corresponding destination during the archival operation. If a LGWR destination fails, the destination automatically reverts to using the archiver (ARCn) process until the error is corrected.

## Examples

The following example shows the LGWR attribute with the LOG_ARCHIVE_DEST_*n* parameter.

```
LOG_ARCHIVE_DEST_3='SERVICE=stby1 LGWR'
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

# DELAY and NODELAY

| Category | DELAY[=*minutes*] | NODELAY |
|---|---|---|
| Datatype of the attribute | Numeric | Keyword |
| Minimum value | 0 minutes | Not applicable |
| Maximum value | Unlimited | Not applicable |
| Default value | 30 minutes | Not applicable |
| Requires attributes ... | SERVICE | Not applicable |
| Conflicts with attributes ... | LOCATION, NODELAY | DELAY |
| Attribute class | ALTER SESSION and ALTER SYSTEM | ALTER SESSION and ALTER SYSTEM |
| Corresponding V$ARCHIVE_DEST column | DELAY_MINS | DELAY_MINS |
| Related V$ARCHIVE_DEST column | DESTINATION | DESTINATION |

## Purpose

When the standby database is in managed recovery mode, redo logs are automatically applied when they arrive from the primary database. However, to protect against the transfer of corrupted or erroneous data from the primary site to the standby site, you might want to create a time lag between archiving a redo log at the primary site and applying that archived redo log at the standby site.

## Defaults

In managed recovery mode, if the DELAY or the NODELAY attribute is not specified with the LOG_ARCHIVE_DEST_*n* parameter, the default is NODELAY.

If the DELAY attribute is specified without a time interval, the default time interval is 30 minutes.

## Attributes

**DELAY[=*minutes*]**
Use the DELAY attribute of the LOG_ARCHIVE_DEST_*n* initialization parameter to specify a time lag for the application of redo logs at the standby site. The DELAY attribute does not affect the transmittal of redo logs to the standby site.

> **Note:** Changes to the DELAY attribute take effect on the next archival operation. In-progress archival operations are not affected.

The DELAY attribute indicates that the archived redo logs at the standby site 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 log is successfully transmitted and archived at the standby site.

You can use the DELAY attribute to set up a configuration where multiple standby databases are maintained in varying degrees of synchronization with the primary database. For example, assume primary database A supports standby databases B, C, and D. Standby database B is set up as the disaster recovery database and therefore has no time lag. Standby database C is set up to protect against logical or physical corruption, and is maintained with a 2-hour delay. Standby database D is maintained with a 4-hour delay and protects against further corruption.

You can override the specified delay interval at the standby site. To immediately apply an archived redo log to the standby database before the time interval expires, use the NODELAY keyword of the RECOVER MANAGED STANDBY DATABASE clause; for example:

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

**NODELAY**
When you specify the NODELAY attribute and the standby database is in managed recovery mode, redo logs are automatically applied when they arrive from the primary database.

**See Also:** Chapter 13, "SQL Statements"

## Examples

The following example shows the DELAY attribute with the LOG_ARCHIVE_DEST_*n* parameter.

```
LOG_ARCHIVE_DEST_3='SERVICE=stby1 DELAY=240'
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

# DEPENDENCY and NODEPENDENCY

| Category | DEPENDENCY=*destination* | NODEPENDENCY |
|---|---|---|
| Datatype of the attribute | String value | Keyword |
| Minimum value | Not applicable | Not applicable |
| Maximum value | Not applicable | Not applicable |
| Default value | Not applicable | Not applicable |
| Requires attributes ... | SERVICE, REGISTER | Not applicable |
| Conflicts with attributes ... | NODEPENDENCY, LOCATION, NOREGISTER, QUOTA_SIZE, QUOTA_USED | DEPENDENCY |
| Attribute class | ALTER SYSTEM only | ALTER SYSTEM only |
| Corresponding V$ARCHIVE_DEST column | DEPENDENCY | DEPENDENCY |
| Related V$ARCHIVE_DEST columns | Not applicable | Not applicable |

## Purpose

Archiving redo logs to a remote database can be defined as being dependent upon the success or failure of an archival operation to another destination. The dependent destination is known as the child destination. The destination on which the child depends is known as the parent destination. Specify the DEPENDENCY attribute with the LOG_ARCHIVE_DEST_*n* parameter to define a local destination, physical standby database, or logical standby database.

## Defaults

If the DEPENDENCY or the NODEPENDENCY attribute is not specified with the LOG_ARCHIVE_DEST_*n* parameter, the default is NODEPENDENCY.

## Attributes

**DEPENDENCY=*destination***

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

- The standby database and the primary database are on the same node. Therefore, the archived redo logs are implicitly accessible to the standby database.
- Clustered file systems provide remote standby databases with access to the primary database archived redo logs.
- Operating system-specific network file systems provide remote standby databases with access to the primary database archived redo logs.
- Mirrored disk technology provides transparent networking support across geographically remote distances.
- Multiple standby databases are on the same remote site, sharing access to common archived redo logs.

In these situations, although a physical archival operation does not occur for the dependent destination, the standby database needs to know the location of the archived redo logs. This allows the standby database to access the archived redo logs when they become available for managed recovery. The DBA must specify a destination as being dependent on the success or failure of a parent destination.

Consider the case of a two-node cluster where a primary node shares access to the destination with the standby node through a mirrored disk device. This configuration, where you maintain a local standby database, is useful for off-loading ad hoc queries and reporting functions.

The primary database archives a redo log locally and, upon successful completion, the archived redo log is immediately available to the standby database for managed recovery. This does not require a physical remote archival operation for the standby destination. In this case, two destinations are used: one for local archiving and another for archiving at the standby site. The standby destination is not valid unless the primary destination succeeds. Therefore, the standby destination has a dependency upon the success or failure of the local destination.

The `DEPENDENCY` attribute has the following restrictions:

- Only standby destinations can have a dependency.
- The parent destination can be either a local or standby destination.
- The `DEPENDENCY` attribute cannot be modified at the session level.
- The `REGISTER` attribute is required.
- The `SERVICE` attribute is required.
- The alternate destination cannot be self-referencing.

When one or more destinations are dependent upon the same parent destination, all attributes of the dependent destinations still apply to that destination. It appears as if the archival operation was performed for each destination, when only one archival operation actually occurred.

Consider, for example, that two standby databases are dependent upon the archived redo log of a parent destination. You can specify different DELAY attributes for each destination, which allows you to maintain a staggered time lag between the primary database and each standby database.

Similarly, a dependent destination can specify an alternate destination, which itself might or might not be dependent on the same parent destination.

---

**Note:** Dependent destinations do not participate in a standby no-data-loss environment.

---

**NODEPENDENCY**
Specifies that there is no dependency on the success or failure of an archival operation to another destination.

## Examples

One reason to use the DEPENDENCY attribute is if the standby database is on the same site as the primary database. Using this configuration, you only need to archive the redo logs once and, because the standby database resides on the local system, it can access the same redo logs. The following is an example of the LOG_ARCHIVE_DEST_*n* parameters in this scenario:

```
# Set up the mandatory local destination:
#
LOG_ARCHIVE_DEST_1='LOCATION=/oracle/dbs/ MANDATORY'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
#
# Set up the dependent standby database that resides on the local system:
#
LOG_ARCHIVE_DEST_2='SERVICE=dest2 DEPENDENCY=LOG_ARCHIVE_DEST_1 OPTIONAL'
LOG_ARCHIVE_DEST_STATE_2=ENABLE
```

Another reason to use the DEPENDENCY attribute is if two standby databases reside on the same system. The parent and child standby databases can be any mix of physical and logical standby databases. The following is an example of this scenario:

```
# Set up the mandatory local destination:
#
LOG_ARCHIVE_DEST_1='LOCATION=/oracle/dbs/ MANDATORY'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
#
# Set up the remote standby database that will receive the logs:
#
LOG_ARCHIVE_DEST_2='SERVICE=dest2 OPTIONAL'
LOG_ARCHIVE_DEST_STATE_2=ENABLE
#
# Set up the remote standby database that resides on the same system as, and is
# dependent on, the first standby database:
#
LOG_ARCHIVE_DEST_3='SERVICE=dest3 DEPENDENCY=LOG_ARCHIVE_DEST_2 OPTIONAL'
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

# LOCATION and SERVICE

| Category | LOCATION=*local_disk_directory* | SERVICE=*net_service_name* |
|---|---|---|
| Datatype of the attribute | String value | String value |
| Minimum value | Not applicable | Not applicable |
| Maximum value | Not applicable | Not applicable |
| Default value | Not applicable | Not applicable |
| Requires attributes... | Not applicable | Not applicable |
| Conflicts with attributes ... | SERVICE, DELAY, DEPENDENCY, REGISTER=*location_format*, NOREGISTER, ASYNC, TEMPLATE, NET_TIMEOUT | LOCATION, QUOTA_USED, QUOTA_SIZE |
| Attribute class | ALTER SESSION and ALTER SYSTEM | ALTER SESSION and ALTER SYSTEM |
| Corresponding V$ARCHIVE_DEST column | DESTINATION | DESTINATION |
| Related V$ARCHIVE_DEST column | TARGET | TARGET |

## Purpose

Each destination *must* either identify a local disk directory or a remotely accessed database.

Use either the LOCATION or the SERVICE attribute to specify a destination where log transport services archive redo logs. For each Data Guard configuration, you must identify at least one local disk directory (LOCATION=*local_disk_directory*) where redo logs are archived. You can specify up to nine additional local or remote destinations. You identify remote destinations by specifying a valid Oracle Net service name (SERVICE=*net_service_name*).

---

**Note:** If you are specifying multiple attributes, specify either the LOCATION or SERVICE attribute on the first line of the initialization parameter file.

---

Use remote archived redo logs to maintain a transactionally consistent copy of the primary database. Do not use locally archived redo logs to maintain a transactionally consistent standby database. However, when you configure log transport services, you must specify at least one local destination. This ensures that the locally archived redo logs are accessible should manual recovery of the primary database be necessary.

To verify the current settings, query the V$ARCHIVE_DEST fixed view:

- The TARGET column of the V$ARCHIVE_DEST fixed view identifies if the destination is local or remote to the primary database.
- The DESTINATION column of the V$ARCHIVE_DEST fixed view identifies the values that were specified for a destination. For example, the destination parameter value specifies the Oracle Net service name identifying the remote Oracle instance where the archived logs are located.

## Defaults

One of these attributes must be specified. There is no default.

---

**Note:** When changing incremental parameters, there are no default attribute values assumed. For example, LOG_ARCHIVE_DEST_1='SERVICE=stby1 LGWR' assumes the SYNC=PARALLEL option. LOG_ARCHIVE_DEST_1='ARCH' fails because SYNC=PARALLEL conflicts with ARCH even though the default for ARCH is SYNC=NOPARALLEL.

---

## Attributes

**LOCATION=*local_disk_directory***

If you use the LOCATION attribute, specify a valid path name for a disk directory on the system that hosts the primary database. Each destination that specifies the LOCATION attribute must identify a unique directory path name. This is the local destination for archiving redo logs.

Local destinations indicate that the archived redo logs are to reside within the file system that is accessible to the primary database. Locally archived redo logs remain physically within the primary database namespace. The destination parameter value specifies the local file system directory path to where the archived redo logs are copied.

**SERVICE=*network_service_name***
If you specify the SERVICE attribute, specify a valid Oracle Net service name.

Archiving redo logs to a remote destination requires a network connection and an Oracle database instance associated with the remote destination to receive the incoming archived redo logs.

The destination parameter value specifies the Oracle Net service name identifying the remote Oracle instance to which the archived redo logs are copied.

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:** *Oracle9i Net Services Administrator's Guide* for details about setting up Oracle Net service names

## Examples

The following example shows the LOCATION attribute with the LOG_ARCHIVE_DEST_*n* parameter:

```
LOG_ARCHIVE_DEST_2='LOCATION=/arc_dest'
LOG_ARCHIVE_DEST_STATE_2=ENABLE
```

The following example shows the SERVICE attribute with the LOG_ARCHIVE_DEST_*n* parameter:

```
LOG_ARCHIVE_DEST_3='SERVICE=stby1'
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

# MANDATORY and OPTIONAL

| Category | MANDATORY | OPTIONAL |
|---|---|---|
| Datatype of the attribute | Keyword | Keyword |
| Minimum value | Not applicable | Not applicable |
| Maximum value | Not applicable | Not applicable |
| Default value | Not applicable | Not applicable |
| Requires attributes ... | Not applicable | Not applicable |
| Conflicts with attributes ... | OPTIONAL | MANDATORY |
| Attribute class | ALTER SESSION and ALTER SYSTEM | ALTER SESSION and ALTER SYSTEM |
| Corresponding V$ARCHIVE_DEST column | BINDING | BINDING |
| Related V$ARCHIVE_DEST columns | Not applicable | Not applicable |

## Purpose

You can specify a policy for reuse of online redo logs using the attributes OPTIONAL or MANDATORY with the LOG_ARCHIVE_DEST_*n* parameter. The archival operation of an optional destination can fail, and the online redo logs are overwritten. If the archival operation of a mandatory destination fails, online redo logs are not overwritten.

The LOG_ARCHIVE_MIN_SUCCEED_DEST=*n* parameter (where *n* is an integer from 1 to 10) specifies the number of destinations that must archive successfully before the log writer process can overwrite the online redo logs. All mandatory destinations and non-standby optional destinations contribute to satisfying the LOG_ARCHIVE_MIN_SUCCEED_DEST=*n* count. For example, you can set the parameter as follows:

```
# Database must archive to at least two locations before
# overwriting the online redo logs.
LOG_ARCHIVE_MIN_SUCCEED_DEST = 2
```

When determining how to set your parameters, note that:

- This attribute does not affect the destination protection mode.
- You must have at least one local destination, which you can declare OPTIONAL or MANDATORY.

  At least one local destination is operationally treated as mandatory, because the minimum value for the LOG_ARCHIVE_MIN_SUCCEED_DEST parameter is 1.
- The failure of any mandatory destination, including a mandatory standby destination, makes the LOG_ARCHIVE_MIN_SUCCEED_DEST parameter irrelevant.
- The LOG_ARCHIVE_MIN_SUCCEED_DEST value cannot be greater than the number of destinations, nor greater than the number of mandatory destinations plus the number of optional local destinations.
- If you defer a mandatory destination, and the online log is overwritten without transferring the redo log to the standby site, then you must transfer the redo log to the standby site manually.

The BINDING column of the V$ARCHIVE_DEST fixed view specifies how failure affects the archival operation.

## Defaults

If the MANDATORY or the OPTIONAL attribute is not specified with the LOG_ARCHIVE_DEST_*n* parameter, the default is OPTIONAL.

At least, one destination must succeed even if all destinations are designated to be optional.

## Attributes

**MANDATORY**
Specifies that archiving to the destination must succeed before the redo log can be made available for reuse.

**OPTIONAL**
Specifies that successful archiving to the destination is not required before the redo log can be made available for reuse. If the "must succeed count" set with the LOG_ARCHIVE_MIN_SUCCEED_DEST parameter is met, the redo log is marked for reuse.

## Examples

The following example shows the MANDATORY attribute with the LOG_ARCHIVE_DEST_*n* parameter.

```
LOG_ARCHIVE_DEST_1='LOCATION=/arc/dest MANDATORY'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_3='SERVICE=stby1 MANDATORY'
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

# MAX_FAILURE and NOMAX_FAILURE

| Category | MAX_FAILURE=*count* | NOMAX_FAILURE |
|---|---|---|
| Datatype of the attribute | Numeric | Keyword |
| Minimum value | 0 | Not applicable |
| Maximum value | None | Not applicable |
| Default value | None | Not applicable |
| Requires attributes ... | REOPEN | Not applicable |
| Conflicts with attributes ... | NOMAX_FAILURE | MAX_FAILURE |
| Dynamically changed by SQL statement . . . | ALTER SYSTEM | ALTER SYSTEM |
| Corresponding V$ARCHIVE_DEST column | MAX_FAILURE | Not applicable |
| Related V$ARCHIVE_DEST columns | FAILURE_COUNT, REOPEN_SECS | Not applicable |

## Purpose

The MAX_FAILURE and the NOMAX_FAILURE attributes allow you to control the number of times log transport services attempts to reestablish communication and resume archival operations with a failed destination.

## Defaults

If you do not specify either the MAX_FAILURE or the NOMAX_FAILURE attribute, the default is NOMAX_FAILURE, which allows an unlimited number of consecutive attempts to transport archived redo logs to the failed destination.

## Attributes

**MAX_FAILURE=*count***

The MAX_FAILURE attribute specifies the maximum number of consecutive times that log transport services attempts archival operations to a failed destination. Using this attribute, you can provide failure resolution for archiving destinations to which you want to retry archival operations after a failure, but not retry indefinitely. When you specify the MAX_FAILURE attribute, you must also set the REOPEN

attribute to specify how often archival operations are retried to the particular destination.

**To Limit the Number of Archival Attempts** If you set both the MAX_FAILURE and REOPEN attributes to nonzero values, log transport services limits the number of archival attempts to the number of times specified by the MAX_FAILURE attribute. Each destination contains an internal failure counter that tracks the number of consecutive archival failures that have occurred. You can view the failure count in the FAILURE_COUNT column of the V$ARCHIVE_DEST fixed view. The related column REOPEN_SECS identifies the REOPEN attribute value.

If an archival operation fails for any reason, the failure count is incremented until:

- The failure no longer occurs and archival operations resume
- The failure count is greater than or equal to the value set for the MAX_FAILURE attribute

> **Note:** Once the failure count for the destination reaches the specified MAX_FAILURE attribute value, the only way to reuse the destination is to modify the MAX_FAILURE attribute value or some other attribute.

- You issue the ALTER SYSTEM SET statement to dynamically change the MAX_FAILURE attribute (or any other destination attribute). The failure count is reset to zero (0) whenever the destination is modified by an ALTER SYSTEM SET statement. This avoids the problem of setting the MAX_FAILURE attribute to a value less than the current failure count value.

> **Note:** Runtime modifications made to the destination, such as changing the QUOTA_USED attribute, do not affect the failure count.

Once the failure count is greater than or equal to the value set for the MAX_FAILURE attribute, the REOPEN attribute value is implicitly set to the value zero (0), which causes log transport services to transport archived redo logs to an alternate destination (defined with the ALTERNATE attribute) on the next archival operation.

**To Attempt Archival Operations Indefinitely** Log transport services attempt to archive to the failed destination indefinitely if you do not specify the MAX_FAILURE attribute (or if you specify MAX_FAILURE=0 or the NOMAX_FAILURE attribute), and

you specify a nonzero value for the REOPEN attribute. If the destination has the MANDATORY attribute, the online redo log is not reclaimable in the event of a repeated failure.

**NOMAX_FAILURE**
Specify the NOMAX_FAILURE attribute to allow an unlimited number of archival attempts to the failed destination.

The NOMAX_FAILURE attribute is equivalent to specifying MAX_FAILURE=0.

## Examples

The following example allows log transport services up to three consecutive archival attempts, tried every 5 seconds, to the arc_dest destination. If the archival operation fails after the third attempt, the destination is treated as if the NOREOPEN attribute was specified.

```
LOG_ARCHIVE_DEST_1='LOCATION=/arc_dest REOPEN=5 MAX_FAILURE=3'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
```

# NET_TIMEOUT and NONET_TIMEOUT

| Category | NET_TIMEOUT=*seconds* | NONET_TIMEOUT |
|---|---|---|
| Datatype of the attribute | Numeric | Not applicable |
| Minimum value | 15 | Not applicable |
| Maximum value | 1200 | Not applicable |
| Default value | Not applicable | Not applicable |
| Requires attributes ... | LGWR with SYNC=PARALLEL<br>or<br>LGWR with ASYNC > 0 | Not applicable |
| Conflicts with attributes ... | ARCH,<br>LOCATION,<br>NONET_TIMEOUT,<br>LGWR with SYNC=NOPARALLEL,<br>LGWR with ASYNC=0 | NET_TIMEOUT |
| Attribute class | ALTER SYSTEM | ALTER SYSTEM |
| Corresponding V$ARCHIVE_DEST column | NET_TIMEOUT | NET_TIMEOUT |
| Related V$ARCHIVE_DEST column | Not applicable | Not applicable |

## Purpose

The NET_TIMEOUT attribute of the LOG_ARCHIVE_DEST_*n* parameter specifies the number of seconds the log writer process on the primary system waits for status from the network server process before terminating the network connection. The NONET_TIMEOUT attribute reverses or undoes the timeout value that you previously specified with the NET_TIMEOUT attribute.

If you do not specify the NET_TIMEOUT attribute (or if you specify the NONET_TIMEOUT attribute, the primary database can potentially stall. To avoid this situation, specify a small, nonzero value for the NET_TIMEOUT attribute so the primary database can continue operation after the user-specified timeout interval expires when waiting for status from the network server.

## Defaults

If the NET_TIMEOUT or the NONET_TIMEOUT attribute is not specified with the LOG_ARCHIVE_DEST_*n* parameter, the default is NONET_TIMEOUT.

## Attributes

**NET_TIMEOUT=*seconds***

The NET_TIMEOUT attribute is used only when the log writer process archives logs using a network server process and when either the ASYNC or the SYNC=PARALLEL attribute is specified. The log writer process waits for the specified amount of time to receive status from the network I/O operation. If the log writer process does not receive acknowledgment within the specified interval, the primary database network connection will be terminated.

In a Data Guard configuration, there are timers that need to be set similarly for each primary-to-standby network connection:

- Set the NET_TIMEOUT attribute on the primary database in the Data Guard configuration.
- Set the Oracle Net EXPIRE_TIME and the TCP/IP keepalive parameters on each standby database in the Data Guard configuration.

Even though the network connection might be terminated on the primary database, the network connection remains active on the standby database until the corresponding TCP/IP network timers expire. For this reason, you need to set the timers comparably on both sides of the network. If the network timers are not set up properly, subsequent attempts by the logwriter process on the primary database to attach to the standby database will fail because the standby database has not yet timed out and the broken network connection still appears to be valid.

**Note:** In general, you should set network timer parameters on the standby system to expire before the timeout period specified by the NET_TIMEOUT attribute on the primary system. For example:

- Set TCP/IP network timers on the standby system, such as the keepalive parameter, lower than the corresponding value set for the NET_TIMEOUT attribute.
- Set the Oracle Net EXPIRE_TIME parameter on the standby system lower than the corresponding value that you set for the NET_TIMEOUT attribute. The EXPIRE_TIME parameter is expressed in minutes.

Without careful coordination of the timeout parameter values on the primary and standby systems, it is possible that the primary system might detect a network problem and disconnect, while the standby database might not recognize the network disconnection if its default network timeout values are too high.

**See Also:** *Oracle9i Net Services Administrator's Guide*

If the log writer process detects a network disconnection, even one that was terminated due to a network timeout, the log writer process automatically tries to reconnect to the standby database. The log writer process does this to resolve network brownouts and false network terminations. In most cases, except when the network is physically broken, the log writer process is able to automatically reconnect to the network.

The log writer process continually attempts to reconnect to the standby database for a period of time that depends on the data protection mode currently set for the primary database. Use the following time estimates as a guideline for how long the log writer process will try to reconnect to the standby database:

- In maximum protection mode, the log writer process tries to reconnect for approximately 5 minutes.
- In maximum availability mode, the log writer process tries to reconnect for approximately 2 minutes.
- In maximum performance mode, the log writer process tries to reconnect for approximately 30 seconds.

**Caution:** Be careful to specify a reasonable value when running in maximum protection mode. A *false* network failure detection might cause the primary instance to shut down.

The actual time the log writer process tries to reconnect depends on the following factors:

- The value of the NET_TIMEOUT attribute, which determines how long it takes to time out the connection.
- The value of the EXPIRE_TIME parameter or keep alive intervals on the standby database, which determine the minimum amount of time that the reconnection will take on the primary database
- The protection mode of the primary database, which determines the maximum amount of time that the reconnection will take.

For example, a primary database operating in the maximum availability protection mode with a NET_TIMEOUT attribute value set to 60 seconds and an EXPIRE_TIME of 1 minute could actually take a minimum of 1 minute to connect or up to 3 minutes to terminate the connection to the standby database.

#### NONET_TIMEOUT

The NONET_TIMEOUT attribute implies that the log writer process waits for the default network timeout interval established for the system. The default network timeout interval differs from system to system. On some systems, the default TCP/IP network timeout can be between 10 and 15 minutes.

## Examples

The following example shows how to specify a 40-second network timeout value on the primary database with the NET_TIMEOUT attribute.

```
LOG_ARCHIVE_DEST_2='SERVICE=stby1 LGWR NET_TIMEOUT=40 SYNC=PARALLEL'
LOG_ARCHIVE_DEST_STATE_2=ENABLE
```

# QUOTA_SIZE and NOQUOTA_SIZE

| Category | QUOTA_SIZE=*blocks* | NOQUOTA_SIZE |
|---|---|---|
| Datatype of the attribute | Numeric | Keyword |
| Minimum value | 0 blocks | Not applicable |
| Maximum value | Unlimited blocks | Not applicable |
| Default value | Not applicable | Not applicable |
| Requires attributes ... | LOCATION | Not applicable |
| Conflicts with attributes ... | NOQUOTA_SIZE, DEPENDENCY, SERVICE | QUOTA_SIZE |
| Attribute class | ALTER SYSTEM only | ALTER SYSTEM only |
| Corresponding V$ARCHIVE_DEST column | QUOTA_SIZE | QUOTA_SIZE |
| Related V$ARCHIVE_DEST column | QUOTA_USED | QUOTA_USED |

## Purpose

The QUOTA_SIZE and the NOQUOTA_SIZE attributes of the LOG_ARCHIVE_DEST_*n* parameter indicate the maximum number of 512-byte blocks of physical storage on a disk device that can be used by a local destination.

## Defaults

If the QUOTA_SIZE or the NOQUOTA_SIZE attribute is not specified with the LOG_ARCHIVE_DEST_*n* parameter, the default is NOQUOTA_SIZE.

## Attributes

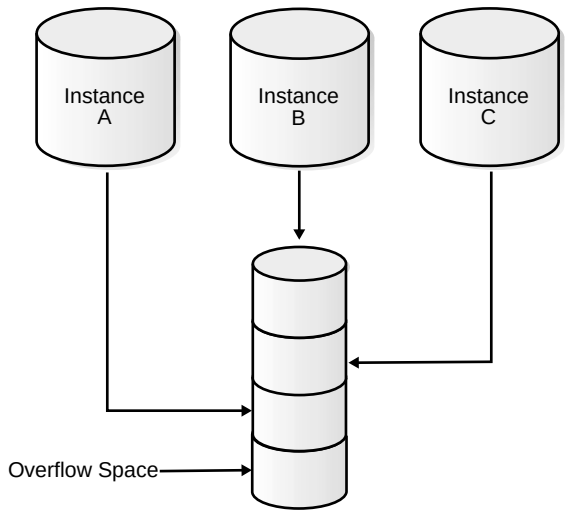
### QUOTA_SIZE=*blocks*

The QUOTA_SIZE attribute indicates the maximum number of 512-byte blocks of physical storage on a disk device that might be used by a local destination. The value is specified in 512-byte blocks even if the physical device uses a different block size. The optional suffix values K, M, and G represent thousand, million, and billion, respectively (the value 1K means 1,000 512-byte blocks).

A local archiving destination can be designated as being able to occupy all or some portion of the physical disk. For example, in a Real Application Clusters environment, a physical archived redo log disk device might be shared by two or more separate nodes (through a clustered file system, such as is available with Sun Clusters). As there is no cross-instance initialization parameter knowledge, none of the Real Application Clusters nodes is aware that the archived redo log physical disk device is shared with other instances. This can lead to significant problems when the destination disk device becomes full; the error is not detected until every instance tries to archive to the already full device. This seriously affects database availability.

For example, consider an 8-gigabyte (GB) disk device `/dev/arc_dest` that is further subdivided into node-specific directories `node_a`, `node_b`, and `node_c`. The DBA could designate that each of these instances is allowed to use a maximum of 2 GB, which would allow an additional 2 GB for other purposes. This scenario is shown in Figure 12–2.

***Figure 12–2 Specifying Disk Quota for a Destination***




No instance uses more than its allotted quota.

The quota is common to all users of the destination, including foreground archival operations, the archiver process, and even the log writer process.

Oracle Corporation highly recommends that the ALTERNATE attribute be used in conjunction with the QUOTA_SIZE attribute. However, this is not required.

**See Also:** ALTERNATE and NOALTERNATE on page 12-9

**NOQUOTA_SIZE**
Use of the NOQUOTA_SIZE attribute, or the QUOTA_SIZE attribute with a value of zero (0), indicates that there is unlimited use of the disk device by this destination; this is the default behavior.

## Examples

The following example shows the QUOTA_SIZE attribute with the LOG_ARCHIVE_DEST_*n* parameter.

```
LOG_ARCHIVE_DEST_4='QUOTA_SIZE=100K'
```

# QUOTA_USED and NOQUOTA_USED

| Category | QUOTA_USED=*blocks* | NOQUOTA_USED |
|---|---|---|
| Datatype of the attribute | Numeric | Keyword |
| Minimum value | 0 blocks | Not applicable |
| Maximum value | Unlimited blocks | Not applicable |
| Default value | Not applicable | Not applicable |
| Requires attributes ... | LOCATION | Not applicable |
| Conflicts with attributes ... | NOQUOTA_USED, DEPENDENCY, SERVICE | QUOTA_USED |
| Attribute class | ALTER SYSTEM only | ALTER SYSTEM only |
| Corresponding V$ARCHIVE_DEST column | QUOTA_USED | QUOTA_USED |
| Related V$ARCHIVE_DEST column | QUOTA_SIZE | QUOTA_SIZE |

## Purpose

The QUOTA_USED and the NOQUOTA_USED attributes of the LOG_ARCHIVE_DEST_*n* parameter identify the number of 512-byte blocks of data that were archived on a specified destination.

## Defaults

If the QUOTA_USED or the NOQUOTA_USED attribute is not specified with the LOG_ARCHIVE_DEST_*n* parameter, the default is NOQUOTA_USED.

The QUOTA_USED attribute has a default value of zero (0) for remote archiving destinations.

## Attributes

**QUOTA_USED=*blocks***

The QUOTA_USED attribute identifies the number of 512-byte blocks of data that were archived on the specified local destination. The value is specified in 512-byte blocks even if the physical device uses a different block size. The optional suffix

values K, M, and G represent thousand, million, and billion, respectively (the value 1K means 1,000 512-byte blocks).

This attribute cannot be modified at the session level.

If you specify a QUOTA_SIZE attribute value greater than zero (0) for a destination, but do not specify a QUOTA_USED attribute value in the database initialization parameter file, the QUOTA_USED attribute value is automatically determined when the database is initially mounted. The QUOTA_USED attribute value defaults to the actual number of blocks residing on the local archiving destination device. If the calculated QUOTA_USED attribute value exceeds the QUOTA_SIZE attribute value, the QUOTA_SIZE attribute value is automatically adjusted to reflect the actual storage used.

This automatic calculation of the QUOTA_USED value applies only to local archiving destinations.

---

**Note:** The runtime value of the QUOTA_USED attribute changes automatically as online redo log archival operations are started. The QUOTA_USED attribute value is automatically pre-allocated against the destination quota size. You do not need to change the value of this attribute.

---

If, at runtime, you dynamically modify the QUOTA_SIZE attribute value, but not the QUOTA_USED attribute value, the QUOTA_USED attribute value is not automatically recalculated.

For local destinations, the QUOTA_USED attribute value is incremented at the start of an archival operation. If the resulting value is greater than the QUOTA_SIZE attribute value, the destination status is changed to FULL, and the destination is rejected before the archival operation begins.

The QUOTA_SIZE and QUOTA_USED attributes are very important because they can be used together to detect a lack of disk space before the archival operation begins.

Consider the case where the QUOTA_SIZE attribute value is 100K and the QUOTA_USED attribute value is 100K also. The destination status is VALID at this point. However, an attempt to archive 1 block results in the QUOTA_USED attribute value being changed to 101K, which exceeds the QUOTA_SIZE attribute value. Therefore, the destination status is changed to FULL, and the destination is rejected before the archival operation begins.

**NOQUOTA_USED**
Specifies that an unlimited number of blocks of data can be archived on a specified destination.

## Examples

Data Guard automatically sets this value. You do not need to change the value of the QUOTA_USED and the NOQUOTA_USED attributes.

# REGISTER and NOREGISTER

| Category | REGISTER | NOREGISTER |
|---|---|---|
| Datatype of the attribute | Keyword | Keyword |
| Minimum value | Not applicable | Not applicable |
| Maximum value | Not applicable | Not applicable |
| Default value | Not applicable | Not applicable |
| Requires attributes ... | Not applicable | SERVICE |
| Conflicts with attributes ... | NOREGISTER, NOALTERNATE | REGISTER, LOCATION |
| Attribute class | ALTER SESSION and ALTER SYSTEM | ALTER SESSION and ALTER SYSTEM |
| Corresponding V$ARCHIVE_DEST column | DESTINATION | DESTINATION |
| Related $ARCHIVE_DEST column | TARGET | TARGET |

## Purpose

The REGISTER and the NOREGISTER attributes of the LOG_ARCHIVE_DEST_*n* parameter indicate if the location of the archived redo log is to be recorded at the destination site.

## Defaults

If the REGISTER or the NOREGISTER attribute is not specified with the LOG_ARCHIVE_DEST_*n* parameter, the default is REGISTER.

## Attributes

### REGISTER

The REGISTER attribute indicates that the location of the archived redo log is to be recorded at the corresponding destination.

For a physical standby destination, the archived redo log filename is recorded in the destination database control file, which is then used by the managed recovery operation.

For a logical standby database, the archived redo log filename is recorded in the tablespace maintained by the logical standby database control file which is then used by SQL apply operations.

The REGISTER attribute implies that the destination is a Data Guard standby database.

By default, the location of the archived redo log, at a remote destination site, is derived from the destination instance initialization parameters STANDBY_ARCHIVE_DEST and LOG_ARCHIVE_FORMAT.

---

**Note:** You can also set the REGISTER attribute by executing the SQL ALTER DATABASE REGISTER LOGFILE *filespec* statement on each standby database. See Section 7.2.2.1 for an example of this SQL statement.

---

**NOREGISTER**

The optional NOREGISTER attribute indicates that the location of the archived redo log is not to be recorded at the corresponding destination. This setting pertains to remote destinations only. The location of each archived redo log is always recorded in the primary database control file.

The NOREGISTER attribute is required if the destination is a standby database that is not part of a Data Guard configuration.

## Examples

The following example shows the REGISTER attribute with the LOG_ARCHIVE_DEST_*n* parameter.

```
LOG_ARCHIVE_DEST_5='REGISTER'
```

# REGISTER=location_format

| Category | REGISTER=*location_format* |
|---|---|
| Datatype of the attribute | String value |
| Minimum value | Not applicable |
| Maximum value | Not applicable |
| Default value | Not applicable |
| Requires attributes ... | DEPENDENCY |
| Conflicts with attributes ... | NOREGISTER, LOCATION, TEMPLATE |
| Attribute class | ALTER SESSION and ALTER SYSTEM |
| Corresponding V$ARCHIVE_DEST column | DESTINATION |
| Related V$ARCHIVE_DEST column | TARGET |

## Purpose

The optional REGISTER=*location_format* attribute is used to specify a filename format template for archived redo logs that is different from the default filename format template defined in the primary and standby database initialization parameter files.

This attribute is for use on physical standby databases only.

## Defaults

There is no default for this attribute.

## Attributes

**REGISTER=*location_format***

The optional REGISTER=*location_format* attribute is used to specify a fully-qualified filename format template for archived redo logs that is different from the default filename format template defined in the primary and standby database initialization parameter files. The default filename format template is a combination

of the database initialization parameters STANDBY_ARCHIVE_DEST and LOG_ARCHIVE_FORMAT.

**See Also:** REGISTER and NOREGISTER on page 12-42

The REGISTER=*location_format* attribute is valid with remote destinations only.

## Examples

The following example shows the REGISTER=*location_format* attribute with the LOG_ARCHIVE_DEST_*n* parameter.

```
LOG_ARCHIVE_DEST_4='REGISTER=/disk1/oracle/oradata/payroll/arc%d_%t_%s.arc'
```

# REOPEN and NOREOPEN

| Category | REOPEN [=*seconds*] | NOREOPEN |
|---|---|---|
| Datatype of the attribute | Numeric | Keyword |
| Minimum value | 0 seconds | Not applicable |
| Maximum value | Unlimited seconds | Not applicable |
| Default value | 300 seconds | Not applicable |
| Requires attributes ... | Not applicable | Not applicable |
| Conflicts with attributes ... | NOREOPEN | REOPEN |
| Attribute class | ALTER SESSION and ALTER SYSTEM | ALTER SESSION and ALTER SYSTEM |
| Corresponding V$ARCHIVE_DEST column | REOPEN_SECS | REOPEN_SECS |
| Related V$ARCHIVE_DEST column | MAX_FAILURE | MAX_FAILURE |

## Purpose

The REOPEN and the NOREOPEN attributes of the LOG_ARCHIVE_DEST_*n* parameter specify the minimum number of seconds before the archiver process (ARC*n*, foreground, or log writer process) should try again to access a previously failed destination. You can turn off the attribute by specifying NOREOPEN.

## Defaults

If the REOPEN or the NOREOPEN attribute is not specified with the LOG_ARCHIVE_DEST_*n* parameter, the default is REOPEN. If the REOPEN attribute is specified without an integer value, the default is 300 seconds.

## Attributes

**REOPEN[=*seconds*]**
REOPEN applies to all errors, not just connection failures. These errors include, but are not limited to, network failures, disk errors, and quota exceptions.

If you specify REOPEN for an OPTIONAL destination, it is still possible for the Oracle database server to overwrite online redo logs even if there is an error. If you specify REOPEN for a MANDATORY destination, log transport services stall the primary database when they cannot successfully archive redo logs. When this situation occurs, consider the following options:

- Change the destination by deferring the destination, specifying the destination as optional, or changing the service.
- Specify an alternate destination.
- Disable the destination.

When you use the REOPEN attribute, note that:

- The archiver or log writer process reopens a destination only when starting an archive operation from the beginning of the log and never during a current operation. Archiving always starts the log copy from the beginning.
- If a value was specified, or the default value was used, for the REOPEN attribute, the archiving process checks if the time of the recorded error plus the REOPEN interval is less than the current time. If it is, the archival operation to that destination is retried.
- You can control the number of times a destination will be retried after a log archiving failure by specifying a value for the MAX_FAILURE=*count* attribute of the LOG_ARCHIVE_DEST_*n* initialization parameter.

**NOREOPEN**

If you specify NOREOPEN, the failed destination remains disabled until:

- You manually reenable the destination.
- You issue an ALTER SYSTEM SET or an ALTER SESSION SET statement with the REOPEN attribute.
- The instance is restarted.

## Examples

The following example shows the REOPEN attribute with the LOG_ARCHIVE_DEST_*n* parameter.

```
LOG_ARCHIVE_DEST_3='SERVICE=stby1 MANDATORY REOPEN=60'
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

# SYNC and ASYNC

| Category | SYNC[=*parallel_option*] | ASYNC[=*blocks*] |
|---|---|---|
| Datatype of the attribute | Keyword | Numeric |
| Minimum value | Not applicable | 0 blocks |
| Maximum value | Not applicable | 20,480 blocks |
| Default value | Not applicable | 2,048 |
| Requires attributes ... | Not applicable | LGWR |
| Conflicts with attributes ... | ASYNC | SYNC, LOCATION, ARCH |
| Attribute class | ALTER SESSION and ALTER SYSTEM | ALTER SYSTEM only |
| Corresponding V$ARCHIVE_DEST column | TRANSMIT_MODE | TRANSMIT_MODE |
| Related V$ARCHIVE_DEST column | Not applicable | ASYNC_BLOCKS |

## Purpose

The SYNC and the ASYNC attributes of the LOG_ARCHIVE_DEST_*n* parameter specify that network I/O operations are to be done synchronously or asynchronously when using the log writer process (LGWR).

**Note:** When the primary database is in one of the three protection modes, standby redo log archiving destinations using the log writer process are automatically placed in SYNC mode.

## Defaults

If the SYNC or ASYNC attribute is not specified, the default is SYNC. If the destination defaults to SYNC, or the SYNC attribute is specified without specifying the PARALLEL qualifier, the default for the PARALLEL qualifier depends on which transmitter process is chosen for the destination. When you specify the LGWR attribute, the default parallel qualifier is PARALLEL. Because the PARALLEL qualifier is not allowed with the ARCH attribute, when you specify the ARCH attribute, the default parallel qualifier is NOPARALLEL.

If the ASYNC attribute is specified without an integer value, the default is 2048 blocks.

## Attributes

**SYNC=PARALLEL**
**SYNC=NOPARALLEL**
The SYNC attribute specifies that network I/O is to be performed synchronously for the destination, which means that once the I/O is initiated, the archiving process waits for the I/O to complete before continuing. The SYNC attribute is one requirement for setting up a no-data-loss environment, because it ensures that the redo records were successfully transmitted to the standby site before continuing.

If the log writer process is defined to be the transmitter to multiple standby destinations that use the SYNC attribute, the user has the option of specifying SYNC=PARALLEL or SYNC=NOPARALLEL for each of those destinations.

- If SYNC=NOPARALLEL is used, the log writer process performs the network I/O to each destination in series. In other words, the log writer process initiates an I/O to the first destination and waits until it completes before initiating the I/O to the next destination. Specifying the SYNC=NOPARALLEL attribute is the same as specifying the ASYNC=0 attribute.
- If SYNC=PARALLEL is used, the network I/O is initiated asynchronously, so that I/O to multiple destinations can be initiated in parallel. However, once the I/O is initiated, the log writer process waits for each I/O operation to complete before continuing. This is, in effect, the same as performing multiple, synchronous I/O operations simultaneously. The use of SYNC=PARALLEL is likely to perform better than SYNC=NOPARALLEL.

Because the PARALLEL and NOPARALLEL qualifiers only make a difference if multiple destinations are involved, Oracle Corporation recommends that all destinations use the same value.

**ASYNC[=*blocks*]**
The ASYNC attribute specifies that network I/O is to be performed asynchronously for the destination. Once the I/O is initiated, the log writer continues processing the next request without waiting for the I/O to complete and without checking the completion status of the I/O. Use of the ASYNC attribute allows standby environments to be maintained with little or no performance effect on the primary database. The optional block count determines the size of the SGA network buffer to be used. In general, the slower the network connection, the larger the block count

should be. Also, specifying the ASYNC=0 attribute is the same as specifying the SYNC=NOPARALLEL attribute.

When you use the ASYNC attribute, there are several events that cause the network I/O to be initiated:

- If the LGWR request exceeds the currently available buffer space, the existing buffer is transmitted to the standby database. The LGWR process stalls until sufficient buffer space can be reclaimed.
- A primary database log switch forces any buffered redo logs to be transmitted to the standby database before the log switch operation completes.
- The primary database is shut down normally. An immediate shutdown of the primary database results in the buffered redo logs being discarded. A standby database shutdown also causes the buffered redo logs to be discarded.
- The primary database has no redo activity for a period of time. The duration of database inactivity is determined by the system and you cannot modified it.
- If the rate of redo generation exceeds the runtime network latency and sufficient space is available, then the LGWR request will be buffered. Otherwise, the existing buffer is transmitted to the standby database. The LGWR process stalls until sufficient buffer space can be reclaimed.

## Examples

The following example shows the SYNC attribute with the LOG_ARCHIVE_DEST_*n* parameter.

```
LOG_ARCHIVE_DEST_3='SERVICE=stby1 LGWR SYNC'
LOG_ARCHIVE_DEST_STATE_3=ENABLE
```

# TEMPLATE and NOTEMPLATE

| Category | TEMPLATE=*filename_template* | NOTEMPLATE |
|---|---|---|
| Datatype of the attribute | String value | Not applicable |
| Minimum value | Not applicable | Not applicable |
| Maximum value | Not applicable | Not applicable |
| Default value | Not applicable | Not applicable |
| Requires attributes ... | SERVICE | Not applicable |
| Conflicts with attributes ... | NOTEMPLATE, LOCATION, REGISTER=*location_format* | TEMPLATE |
| Attribute class | ALTER SESSION and ALTER SYSTEM | ALTER SESSION and ALTER SYSTEM |
| Corresponding V$ARCHIVE_DEST column | REMOTE_TEMPLATE | REMOTE_TEMPLATE |
| Related V$ARCHIVE_DEST column | REGISTER | REGISTER |

## Purpose

The TEMPLATE and the NOTEMPLATE attributes of the LOG_ARCHIVE_DEST_*n* parameter define a directory specification and format template for archived redo logs at the standby destination. You can specify this attribute in either the primary or standby initialization parameter file, but the attribute applies only to the database role that is archiving.

The TEMPLATE attribute overrides the STANDBY_ARCHIVE_DEST and LOG_ARCHIVE_FORMAT initialization parameter settings at the remote archive destination.

The TEMPLATE and NOTEMPLATE attributes are valid only with remote destinations.

**Note:** If used on a LGWR destination, rearchival by the ARC*n* process does not use the TEMPLATE specification. This is important for protected destinations.

## Defaults

There is no default for this attribute.

## Attributes

**TEMPLATE=*filename_template***
Use the optional TEMPLATE attribute to define a directory specification and format for archived redo logs at the standby destination. The definition is used to generate a filename that is different from the default filename format defined by the STANDBY_ARCHIVE_DEST and LOG_ARCHIVE_FORMAT initialization parameters at the standby destination.

The *filename_template* value of the TEMPLATE attribute must contain a thread or sequence number directive. The following table provides a definition for each directive.

| Thread or Sequence Number Directive | Description |
|---|---|
| %a | Substitute the database activation ID. |
| %A | Substitute the database activation ID zero-filled. |
| %d | Substitute the database ID. |
| %D | Substitute the database ID zero-filled. |
| %t | Substitute the instance thread number. |
| %T | Substitute the instance thread number zero-filled. |
| %s | Substitute the log file sequence number. |
| %S | Substitute the log file sequence number zero-filled. |

The *filename_template* value is transmitted to the standby destination, where it is translated and validated before creating the filename.

If you do not specify the TEMPLATE attribute, the setting is the same as REGISTER.

**NOTEMPLATE**
Use the optional NOTEMPLATE attribute to allow the filename format template defined by the STANDBY_ARCHIVE_DEST and LOG_ARCHIVE_FORMAT initialization parameters to take effect.

## Examples

The following example shows the TEMPLATE attribute with the LOG_ARCHIVE_DEST_*n* parameter.

```
LOG_ARCHIVE_DEST_1='SERVICE=stby1 MANDATORY REOPEN=5
    TEMPLATE=/usr/oracle/prmy1/p1_%t_%s.dbf'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
```

prmy1 archives redo logs at the remote destination. stby1 is located in the directory /usr/oracle/prmy1 with the filename format of p1_<thread#>_<sequence#>.dbf.

## 12.4 Attribute Compatibility for Archive Destinations

The LOG_ARCHIVE_DEST_*n* initialization parameter has many attributes. Some of these attributes conflict with each other. Some of the attributes require that other attributes are also defined. Table 12–2 lists the supported attributes and the requirements associated with each one.

***Table 12–2 LOG_ARCHIVE_DEST_n Attribute Compatibility***

| Attribute | Requires... | Conflicts with... |
|---|---|---|
| AFFIRM | Not applicable | NOAFFIRM |
| NOAFFIRM | Not applicable | AFFIRM |
| ALTERNATE=*destination* | Not applicable | NOALTERNATE |
| NOALTERNATE | Not applicable | ALTERNATE |
| ARCH | Not applicable | LGWR<br>ASYNC<br>NET_TIMEOUT |
| ASYNC[=*blocks*] | LGWR | SYNC<br>LOCATION<br>ARCH |
| DELAY | SERVICE | LOCATION<br>NODELAY |
| NODELAY | Not applicable | DELAY |
| DEPENDENCY | SERVICE<br>REGISTER | LOCATION<br>NODEPENDENCY<br>NOREGISTER<br>QUOTA_SIZE<br>QUOTA_USED |
| NODEPENDENCY | Not applicable | DEPENDENCY |
| LGWR | Not applicable | ARCH |
| LOCATION | Not applicable | SERVICE<br>DEPENDENCY<br>REGISTER=*location_format*<br>NOREGISTER<br>DELAY<br>ASYNC<br>NET_TIMEOUT<br>TEMPLATE |

***Table 12–2 LOG_ARCHIVE_DEST_n Attribute Compatibility***

| Attribute | Requires... | Conflicts with... |
|---|---|---|
| MANDATORY | Not applicable | OPTIONAL |
| MAX_FAILURE | REOPEN | NOMAX_FAILURE |
| NOMAX_FAILURE | Not applicable | MAX_FAILURE |
| NET_TIMEOUT | LGWR with SYNC=PARALLEL<br>or<br>LGWR with ASYNC > 0 | ARCH<br>LOCATION<br>NONET_TIMEOUT<br>LGWR with SYNC=NOPARALLEL<br>LGWR with ASYNC=0 |
| NONET_TIMEOUT | Not applicable | NET_TIMEOUT |
| OPTIONAL | Not applicable | MANDATORY |
| QUOTA_SIZE | LOCATION | DEPENDENCY<br>SERVICE<br>NOQUOTA_SIZE |
| NOQUOTA_SIZE | Not applicable | QUOTA_SIZE |
| QUOTA_USED | LOCATION | DEPENDENCY<br>SERVICE<br>NOQUOTA_USED |
| NOQUOTA_USED | Not applicable | QUOTA_USED |
| REGISTER | Not applicable | NOALTERNATE<br>NOREGISTER |
| NOREGISTER | SERVICE | LOCATION<br>REGISTER |
| REGISTER=<br>*location_format* | DEPENDENCY | LOCATION<br>NOREGISTER<br>TEMPLATE |
| REOPEN | Not applicable | NOREOPEN |
| NOREOPEN | Not applicable | REOPEN |
| SERVICE | Not applicable | LOCATION<br>QUOTA_USED<br>QUOTA_SIZE |
| SYNC[=<br>*parallel_option*] | Not applicable | ASYNC |

***Table 12–2 LOG_ARCHIVE_DEST_n Attribute Compatibility***

| Attribute | Requires... | Conflicts with... |
|---|---|---|
| TEMPLATE | SERVICE | NOTEMPLATE<br>LOCATION<br>REGISTER=*location_format* |
| NOTEMPLATE | Not applicable | TEMPLATE |

# 13

# SQL Statements

This chapter summarizes the SQL statements that are useful for performing operations on standby databases in a Data Guard environment. These include:

```
ALTER DATABASE ACTIVATE STANDBY DATABASE
ALTER DATABASE ADD [STANDBY] LOGFILE
ALTER DATABASE ADD [STANDBY] LOGFILE MEMBER
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA
ALTER DATABASE COMMIT TO SWITCHOVER
ALTER DATABASE CREATE STANDBY CONTROLFILE AS
ALTER DATABASE DROP [STANDBY] LOGFILE
ALTER DATABASE DROP [STANDBY] LOGFILE MEMBER
ALTER DATABASE [NO]FORCE LOGGING
ALTER DATABASE MOUNT STANDBY DATABASE
ALTER DATABASE OPEN READ ONLY
ALTER DATABASE RECOVER MANAGED STANDBY DATABASE
ALTER DATABASE REGISTER LOGFILE
ALTER DATABASE SET STANDBY DATABASE TO MAXIMIZE {PROTECTION | AVAILABILITY | PERFORMANCE}
ALTER DATABASE START LOGICAL STANDBY APPLY
ALTER DATABASE {STOP | ABORT} LOGICAL STANDBY APPLY
```

**See Also:** *Oracle9i SQL Reference* for additional information about these and other SQL statements

## 13.1 ALTER DATABASE ACTIVATE STANDBY DATABASE

This statement performs a forced failover operation, in which the primary database is removed from the Data Guard environment and a standby database assumes the primary database role. The standby database must be mounted before it can be activated with this statement. The SQL statement syntax is:

```
ALTER DATABASE ACTIVATE [PHYSICAL | LOGICAL] STANDBY DATABASE [SKIP [STANDBY LOGFILE]];
```

Table 13–1 describes the keywords for this statement.

*Table 13–1 Keywords for the ACTIVATE STANDBY DATABASE Clause*

| Keyword | Description |
|---|---|
| PHYSICAL | Activates a physical standby database. This is the default. |
| LOGICAL | Activates a logical standby database. If you have more than one logical standby database, you should first ensure that the same log data is available on all the logical standby sites. |
| SKIP [STANDBY LOGFILE]<br>(Physical standby databases only) | Forces the failover operation to proceed even if standby redo logs contain data that could be recovered using the RECOVER MANAGED STANDBY DATABASE FINISH clause. Using the SKIP [STANDBY LOGFILE] clause indicates that it is acceptable to discard the contents of the standby redo log. |

**Note:** Oracle Corporation recommends that you perform a failover operation using the ALTER DATABASE RECOVER MANAGED STANDBY DATABASE statement with the FINISH or FINISH SKIP keywords rather than a forced failover operation whenever possible. A forced failover operation renders other standby databases that are not participating in the failover operation unusable as standby databases to the newly activated primary database.

## 13.2 ALTER DATABASE ADD [STANDBY] LOGFILE

This statement adds one or more redo log groups to the specified thread, making the logs available to the instance assigned the thread. The SQL statement syntax is:

```
ALTER DATABASE ADD [STANDBY] LOGFILE [THREAD integer] [GROUP integer] [REUSE] SIZE
filespec;
```

Table 13–2 describes the keywords for this statement.

*Table 13–2 Keywords for the ADD STANDBY LOGFILE Clause*

| Keyword | Description |
|---|---|
| STANDBY | Indicates that the redo log created is for use by standby databases only. |
| THREAD *integer* | Applicable only if you are using the Real Application Clusters option in parallel mode. The *integer* variable is the thread number. |

*Table 13–2 Keywords for the ADD STANDBY LOGFILE Clause*

| Keyword | Description |
|---|---|
| GROUP *integer* | Uniquely identifies the redo log group among all groups in all threads and can range from 1 to the MAXLOGFILES value. You cannot add multiple redo log groups having the same GROUP value. |
| *filespec* | Specifies a redo log group containing one or more members. |
| REUSE | If the log already exists, you can specify REUSE to allow Data Guard to overwrite the header information in the log. |
| SIZE | Specify the size of the log in bytes. Use K or M to specify the size in kilobytes or megabytes. |

See Section 5.3.3.3 for more information about this SQL statement.

# 13.3 ALTER DATABASE ADD [STANDBY] LOGFILE MEMBER

This statement adds new members to existing redo log groups. The SQL statement syntax is:

```
ALTER DATABASE ADD [STANDBY] LOGFILE MEMBER 'filename' [REUSE] TO logfile-descriptor;
```

Table 13–3 describes the keywords for this statement.

*Table 13–3 Keywords for the ADD STANDBY LOGFILE MEMBER Clause*

| Keyword | Description |
|---|---|
| STANDBY | Indicates that the log member is for use only by a standby database. The STANDBY keyword is not required, but you can use it for symmetry in scripts, if necessary. |
| LOGFILE MEMBER *'filename'* | Adds new members to existing redo log groups. Each new member is specified by *'filename'*. |
| REUSE | If the file specified by *'filename'* already exists, the log must be the same size as the other group members, and you must specify REUSE to allow the Oracle server to overwrite the existing log. If the log does not already exist, the Oracle server creates a log of the correct size. |

*Table 13–3 Keywords for the ADD STANDBY LOGFILE MEMBER Clause*

| Keyword | Description |
|---|---|
| *logfile_descriptor* | Specify an existing log group using either of these ways:<br>■ Specify the GROUP *integer* parameter, where *integer* is the number of the existing log group. If this log group was added for standby database use, all members of the log group will be used only for standby databases.<br>■ List the full filename specification for each member of the redo log group. Specify the filename according to the conventions for your operating system. |

See Section 5.3.3.4 for more information about this SQL statement.

# 13.4 ALTER DATABASE ADD SUPPLEMENTAL LOG DATA

This statement is for logical standby databases only.

You must enable full supplemental logging before you create the logical standby database. This is 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 UNIQUE INDEX COLUMNS keyword. The SQL statement syntax is:

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA {PRIMARY KEY | UNIQUE INDEX} COLUMNS;
```

Table 13–4 describes the keywords for this statement.

*Table 13–4 Keywords for the ADD SUPPLEMENTAL LOG DATA Clause*

| Keyword | Description |
|---|---|
| PRIMARY KEY | Ensures, for all tables with a primary key, that all columns of the primary key are placed into the redo log whenever an update operation is performed. If no primary key is defined, Oracle software places into the redo log a set of columns that uniquely identifies the row. |
| UNIQUE INDEX | Ensures, for all tables with a unique index, that if any unique index columns are modified, all other columns belonging to the unique index are also placed into the redo log. |

See Section 4.1.6 for more information about this SQL statement.

# 13.5 ALTER DATABASE COMMIT TO SWITCHOVER

Use this statement to perform a switchover operation to change the current primary database to the standby database role and to change one standby database to the primary database role. The SQL statement clauses you specify differ depending on if you issue the statement on the primary database, a physical standby database, or a logical standby database:

- On the primary database that you want to change to run in a physical standby database role, use the following SQL statement syntax:

  ```
  ALTER DATABASE COMMIT TO SWITCHOVER TO PHYSICAL STANDBY [[WITH | WITHOUT]
  SESSION SHUTDOWN ] [WAIT | NOWAIT];
  ```

- On the primary database that you want to change to run in a logical standby database role, use the following SQL statement syntax:

  ```
  ALTER DATABASE COMMIT TO SWITCHOVER TO LOGICAL STANDBY [WAIT | NOWAIT];
  ```

- On a physical standby database that you want to change to run in the primary database role, use the following SQL statement syntax:

  ```
  ALTER DATABASE COMMIT TO SWITCHOVER TO PRIMARY [[WITH | WITHOUT] SESSION
  SHUTDOWN ] [WAIT | NOWAIT];
  ```

- On a logical standby database that you want to change to run in the primary database role, use the following SQL statement syntax:

  ```
  ALTER DATABASE COMMIT TO SWITCHOVER TO PRIMARY [WAIT | NOWAIT];
  ```

Table 13–5 describes the keywords for this statement.

***Table 13–5 Keywords for the COMMIT TO SWITCHOVER Clause***

| Keyword | Description |
|---|---|
| `COMMIT TO SWITCHOVER TO PHYSICAL STANDBY` | Transitions the primary database to run in the role of a physical standby database.The physical standby database must be mounted and can be open in `READ ONLY` mode. |
| `COMMIT TO SWITCHOVER TO LOGICAL STANDBY` | Transitions the primary database to run in the role of a logical standby database. This option must be followed by an `ALTER DATABASE START LOGICAL STANDBY APPLY` statement. |

*Table 13–5 Keywords for the COMMIT TO SWITCHOVER Clause*

| Keyword | Description |
|---|---|
| `COMMIT TO SWITCHOVER TO [PHYSICAL \| LOGICAL] PRIMARY` | Transitions the standby database to run in the primary database role. For physical standby databases only, the standby database must be mounted and can be open in `READ ONLY` mode. You can specify the `PHYSICAL` or `LOGICAL` parameters for symmetry (in scripts, for example), but these keywords are not required. |
| `WITH SESSION SHUTDOWN`<br>(Physical standby databases only) | Shuts down any open application sessions and rolls back uncommitted transactions as part of the execution of this statement.<br>Logical standby databases do not support the `WITH SESSION SHUTDOWN` option. |
| `WITHOUT SESSION SHUTDOWN`<br>(Physical standby databases only) | Causes the switchover operation to fail if any application sessions are open. This is the default.<br>Logical standby databases do not support the `WITHOUT SESSION SHUTDOWN` option. |
| `WAIT` | Waits for the completion of the switchover operation before returning control to the user. This is the default. |
| `NOWAIT` | Returns control to the user before the switchover operation is complete. |

See Section 7.2 and Section 7.3 for additional information about this SQL statement.

## 13.6 ALTER DATABASE CREATE STANDBY CONTROLFILE AS

This statement is for physical standby databases only.

This statement creates a standby control file. Issue this statement on the primary database. The SQL statement syntax is:

```
ALTER DATABASE CREATE STANDBY CONTROLFILE AS 'filename' [REUSE];
```

Table 13–6 describes the keywords for this statement.

*Table 13–6 Keywords for the CREATE STANDBY CONTROLFILE AS Clause*

| Keyword | Description |
|---|---|
| `CONTROLFILE AS 'filename'` | Specifies the name of the control file to be created and used to maintain a standby database. |

*Table 13–6 Keywords for the CREATE STANDBY CONTROLFILE AS Clause*

| Keyword | Description |
|---|---|
| REUSE | If the control file specified by the *'filename'* parameter already exists, you must specify REUSE to allow the Oracle server to overwrite the existing file. |

See Section 3.2.3 for more information about this SQL statement.

# 13.7 ALTER DATABASE DROP [STANDBY] LOGFILE

This clause drops all members of a redo log group. The SQL statement syntax is:

```
ALTER DATABASE DROP [STANDBY] LOGFILE logfile_descriptor;
```

Table 13–7 describes the keywords for this statement.

*Table 13–7 Keywords for the DROP [STANDBY] LOGFILE Clause*

| Keyword | Description |
|---|---|
| STANDBY | Drops all members of a redo log group. You can specify STANDBY for symmetry, but this keyword is not required. |
| *logfile_descriptor* | Specify an existing redo log group using either of these ways:<br>■ Specify the GROUP *integer* parameter, where *integer* is the number of the existing log group.<br>■ List the full filename specification for the redo log group. Specify the filename according to the conventions for your operating system. |

See Section 8.4.4 for an example using this SQL statement.

# 13.8 ALTER DATABASE DROP [STANDBY] LOGFILE MEMBER

This statement drops one or more redo log members. The SQL statement syntax is:

```
ALTER DATABASE DROP [STANDBY] LOGFILE MEMBER 'filename';
```

Table 13–8 describes the keywords for this statement.

*Table 13–8 Keywords for the DROP LOGFILE MEMBER Clause*

| Keyword | Description |
|---|---|
| STANDBY | Drops one or more standby redo log members. You can specify STANDBY for symmetry, but this keyword is not required. |
| *'filename'* | Specifies filenames, separated by commas, for one or more log members. Each *filename* must specify the fully-qualified file specification according to the conventions for your operating system. |

## 13.9 ALTER DATABASE [NO]FORCE LOGGING

Controls whether or not the Oracle database server logs all changes in the database except for changes to temporary tablespaces and temporary segments. The [NO]FORCE LOGGING clause is:

- Required for physical standby databases to prevent inconsistent standby databases
- Recommended for logical standby databases to ensure data availability at the standby database

> **Note:** Oracle Corporation recommends setting the FORCE LOGGING clause before performing the backup operation to create the standby database and remaining in force logging mode for as long as the standby database is active. Also, to prevent performance degradation when in force logging mode, the database should be running in ARCHIVELOG mode.

The primary database must be mounted but not open when you issue this statement. The SQL statement syntax is:

```
ALTER DATABASE [NO]FORCE LOGGING;
```

Table 13–9 describes the keywords for this statement.

*Table 13–9 Keywords for the [NO]FORCE LOGGING Clause*

| Keyword | Description |
|---|---|
| FORCE LOGGING | Logs all changes in the database except for changes in temporary tablespaces and temporary segments. This setting takes precedence over and is independent of any NOLOGGING or FORCE LOGGING settings you specify for individual tablespaces and any NOLOGGING setting you specify for individual database objects. All ongoing, unlogged operations must finish before forced logging can begin. |
| NOFORCE LOGGING | Cancels the force logging mode. NOFORCE LOGGING is the default. |

## 13.10 ALTER DATABASE MOUNT STANDBY DATABASE

Mounts a physical standby database, allowing the standby instance to receive archived redo logs from the primary instance. The SQL statement syntax is:

```
ALTER DATABASE MOUNT STANDBY DATABASE;
```

## 13.11 ALTER DATABASE OPEN READ ONLY

This statement is required for physical standby databases. It can be used for logical standby databases.

Opens a physical standby database in read-only mode. This SQL statement restricts users to read-only transactions, preventing them from generating redo logs. You can use this clause to make a physical standby database available for queries, even while archive logs are being copied from the primary database site.

You must mount the physical standby database before you can open it. The SQL statement syntax is:

```
ALTER DATABASE OPEN READ ONLY;
```

See Section 8.2.2 for more information about this SQL statement.

## 13.12 ALTER DATABASE RECOVER MANAGED STANDBY DATABASE

This statement is for physical standby databases only.

Use this statement to start, control, and cancel managed recovery operations and log apply services for physical standby databases. You can use the RECOVER MANAGED STANDBY DATABASE clause on a database that is mounted, open, or

closed. Although this SQL statement does not require any additional clauses, it provides many options to help you control the managed recovery process. The SQL statement syntax is:

```
ALTER DATABASE RECOVER MANAGED STANDBY DATABASE [ startup_clause | modify_clause | cancel_
clause ];
```

### *startup_clause*

When you start managed recovery operations, you can start log apply services in a foreground or a background session:

- To start a foreground session, the SQL statement syntax is:

  ```
  ALTER DATABASE RECOVER MANAGED STANDBY DATABASE [TIMEOUT | NO TIMEOUT];
  ```

- To start a background session, the SQL statement syntax is:

```
ALTER DATABASE RECOVER MANAGED STANDBY DATABASE DISCONNECT [FROM SESSION] [NO TIMEOUT];
```

### *modify_clause*

The RECOVER MANAGED STANDBY DATABASE clause provides a wealth of options for controlling the managed recovery process, switchover operations, and failover operations. These keywords work the same whether managed recovery operations were started in a foreground or a background session, with the exception of some particular failover and switchover operations.

Keywords can be placed in any order in the SQL statement except when you start a failover operation using the FINISH keyword. This keyword must be specified last in the SQL statement.

```
ALTER DATABASE RECOVER MANAGED STANDBY DATABASE [
[ NO TIMEOUT | TIMEOUT [integer] ]
[ NODELAY | DELAY [integer] ]
[ DEFAULT DELAY ]
[ NO EXPIRE | EXPIRE [integer] ]
[ NEXT [integer] ]
[ NOPARALLEL | PARALLEL [integer]]
[ THROUGH { ALL | NEXT | LAST } SWITCHOVER ]
[ THROUGH ALL ARCHIVELOG  [ THREAD n ] SEQUENCE n ]
[ FINISH [ SKIP [STANDBY LOGFILE] [NOWAIT | WAIT] ] ]
]
```

### *cancel_clause*

To stop a managed recovery session, the SQL statement syntax is:

```
ALTER DATABASE RECOVER MANAGED STANDBY DATABASE CANCEL [IMMEDIATE] [NOWAIT];
```

Table 13–10 describes all of the keywords.

***Table 13–10 Keywords for the RECOVER MANAGED STANDBY DATABASE Clause***

| Keyword | Description | Incompatible with |
|---|---|---|
| CANCEL [IMMEDIATE] [NOWAIT] | Terminates managed recovery. By default, log apply services will finish applying the current archived redo log before terminating.<br>Specify IMMEDIATE to terminate managed recovery either before reading another block from the archived redo log or before opening the next archived redo log, whichever occurs first.<br>Specify NOWAIT to return control to the process that issued the CANCEL statement without waiting for the managed recovery process to terminate. | All other keywords |
| DELAY *integer* | Specifies an absolute delay interval (in minutes) that log apply services will wait before applying the individual archived redo logs. The apply delay interval begins once the archived redo logs are selected for recovery. | CANCEL, FINISH, NODELAY |
| DEFAULT DELAY | Reverts the delay interval to the number of minutes that was specified in the LOG_ARCHIVE_DEST_*n* initialization parameter on the primary database, if any. | CANCEL, DELAY, FINISH, NODELAY |
| DISCONNECT [FROM SESSION] | Starts the managed recovery process (MRP) and log apply services in a background server process. | CANCEL, TIMEOUT |
| NEXT *integer* | Specifies the number of delayed archived redo logs that log apply services should apply as soon as possible after log transport services archive them. | CANCEL, FINISH |
| EXPIRE *integer* | Specifies the number of minutes, relative to the current time, after which the managed recovery operation will automatically terminate. Log apply services will finish applying the current archived redo log before stopping. | CANCEL, FINISH, NO EXPIRE |
| FINISH<br>[SKIP [STANDBY LOGFILE]] [NOWAIT \| WAIT] | Invokes a failover operation that first applies all available archived redo logs and then recovers available standby redo logs.<br>Specify SKIP [STANDBY LOGFILE] to indicate that it is acceptable to skip applying the contents of the standby redo logs.<br>Specify NOWAIT to return control to the foreground process before the recovery completes.<br>Specify WAIT to return control after recovery completes. | CANCEL, DELAY, EXPIRE, NEXT, THROUGH..., TIMEOUT |
| NODELAY | Disables a previously specified DELAY option so that log apply services will apply the archived redo logs to the standby database without delay. | CANCEL, DELAY |

***Table 13–10 Keywords for the RECOVER MANAGED STANDBY DATABASE Clause***

| Keyword | Description | Incompatible with |
|---|---|---|
| NO EXPIRE | Disables a previously specified EXPIRE option. | CANCEL, EXPIRE |
| NO TIMEOUT | Disables a previously specified TIMEOUT option. | CANCEL |
| NOPARALLEL | Disables a previously specified PARALLEL option so that log apply services use a single process to apply all of the archived redo logs sequentially. This is the default. | CANCEL |
| PARALLEL [*integer*] | Starts additional parallel recovery processes to spread the workload of applying the archived redo logs simultaneously to the standby datafiles. By default, Oracle software selects a number of parallel processes that equal the number of CPUs available on all active instances times the value of the PARALLEL_THREADS_PER_CPU initialization parameter. However, you can specify *integer* to indicate the number of parallel processes to use in the parallel operation. Each parallel thread can use one or two parallel execution servers. | CANCEL |
| THROUGH [THREAD *n*] SEQUENCE *n* | Specifies the thread number and sequence number of the archived redo log through which you want to recover. Once the specified archived redo log is applied, managed recovery terminates. The THREAD *n* keyword is optional. If you do not specify THREAD *n*, it defaults to thread 1. | CANCEL, FINISH |

*Table 13–10 Keywords for the RECOVER MANAGED STANDBY DATABASE Clause*

| Keyword | Description | Incompatible with |
|---|---|---|
| THROUGH ALL ARCHIVELOG | Specifies the default behavior for managed recovery mode, which is to continue managed recovery until it is explicitly stopped. This clause is useful to alter a managed recovery that is currently running with a THROUGH THREAD *n* SEQUENCE *n* keyword so that it does not stop after applying the specified archived redo log. | CANCEL, FINISH |
| THROUGH {ALL\|NEXT\|LAST} SWITCHOVER | Keeps log apply services actively applying archived redo logs received from the new primary database after a switchover operation. (By default, log apply services stop after encountering a switchover (end-of-redo) marker within an archived redo log.)<br>ALL - Continues managed recovery until you explicitly cancel it. Managed recovery continues through (ignoring) all switchover (end-of-redo) indicators. The ALL option is useful when there are other standby databases that are not participating in the switchover operation, and you do not want to stop and restart the recovery process on each one.<br>NEXT - Stops managed recovery at the first switchover (end-of-redo) indicator encountered. This is the default behavior.<br>LAST - Continues managed recovery through all switchover (end-of-redo) indicators, stopping only when an end-of-redo marker is encountered in the last archived redo log received. | CANCEL, FINISH |
| TIMEOUT *integer* | Specifies the number of minutes that the managed recovery process waits for the next archived redo log to arrive from the primary database. If another log does not arrive within the specified time, log apply services automatically stop.<br>Specify TIMEOUT only when starting a managed recovery in a foreground session. | CANCEL, DISCONNECT, FINISH |

**See Also:** Section 6.2.2 for complete information about controlling log apply services and the managed recovery process.

## 13.13 ALTER DATABASE REGISTER LOGFILE

This clause allows the registration of manually archived redo logs. The SQL statement syntax is:

```
ALTER DATABASE REGISTER [OR REPLACE] [PHYSICAL | LOGICAL] LOGFILE filespec;
```

Table 13–11 describes the keywords for this statement.

*Table 13–11 Keywords for the REGISTER LOGFILE Clause*

| Keyword | Description |
|---|---|
| `OR REPLACE` | Allows updates to an existing archived redo log entry for the standby database (for example, when the location or file specification for the archived redo log changes). The SCNs of the entries must match exactly, and the original entry must have been created by log transport services. |
| `PHYSICAL` | Indicates that the archived redo log will be registered in the control file for the physical standby database. |
| `LOGICAL` | Indicates that the archived redo log will be registered in the dictionary for the logical standby database. |
| `LOGFILE` *`filespec`* | Specifies a redo log group containing one or more members. Each new member is specified by *filespec*. |

See Section 7.2.2.1 for an example using this SQL statement.

## 13.14 ALTER DATABASE SET STANDBY DATABASE TO MAXIMIZE {PROTECTION | AVAILABILITY | PERFORMANCE}

Use this statement to specify the level of protection for the data in your database environment. Using one of these protection levels, you can protect the primary database against data loss and data divergence. The SQL statement syntax is:

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

You execute this statement on the primary database, which must be stopped and in the mount state. Table 13–12 describes the keywords for this statement.

*Table 13–12 Keywords for the SET STANDBY TO MAXIMIZE Clause*

| Keyword | Description |
|---|---|
| `PROTECTION`<br>(Physical standby databases only) | Offers the highest level of data protection. A transaction does not commit until all data needed to recover that transaction is written to at least one physical standby database that is configured to use the `SYNC` log transport mode. If the primary database is unable to write the redo records to at least one such standby database, the primary database is shut down. This mode guarantees no data loss, but it has the highest potential impact on the performance and availability of the primary database. |

*Table 13–12 Keywords for the SET STANDBY TO MAXIMIZE Clause*

| Keyword | Description |
|---|---|
| AVAILABILITY | Offers the next highest level of data protection. This mode guarantees there will be no data loss between the primary site and at least one standby site in the configuration unless a primary database failure occurs before recovery from a network outage. Then, no data is lost up to the last transaction that was shipped to the site. (Transactions that continued on the primary site after the network went down could be lost.) Unlike maximum protection mode, the primary database will not shut down if it is unable to write the redo records to at least one such standby database. Instead, the protection will be lowered to maximum performance mode until the situation is corrected and the standby database catches up with the primary database. This mode guarantees no data loss unless the primary database fails while in maximum performance mode. Maximum availability mode provides the highest level of data protection that is possible without affecting the availability of the primary database. |
| PERFORMANCE | This is the default protection mode. A primary database transaction commits before the data needed to recover the transaction is written to a standby database. Therefore, some data might be lost if the primary database fails, and you are unable to recover the redo records from the primary database. This mode provides the highest level of data protection that is possible without affecting the performance of the primary database. |

See Section 5.2 for additional information about the data protection modes.

## 13.15 ALTER DATABASE START LOGICAL STANDBY APPLY

This statement is for logical standby databases only.

This statement starts log apply services on the logical standby database. The SQL statement syntax is:

```
ALTER DATABASE START LOGICAL STANDBY APPLY [INITIAL [scn-value] ] [NEW PRIMARY dblink];
```

Table 13–13 describes the keywords for this statement.

*Table 13–13 Keywords for the START LOGICAL STANDBY APPLY Clause*

| Keyword | Description |
|---|---|
| `INITIAL [`*`scn-value`*`]` | Specify this keyword the first time you apply the logs to the logical standby database. It recovers the database to a transaction-consistent state immediately before the system change number (SCN) specified by an integer. |
| `NEW PRIMARY` *`dblink`* | Starts log apply services after a database switchover takes place. This statement ensures that all transactions in the archived redo logs are properly applied to the logical standby database. Specify this keyword after the `ALTER DATABASE COMMIT TO SWITCHOVER TO LOGICAL STANDBY` statement or when a logical standby database has completed processing logs from one primary database and a new database becomes the primary database. It uses the database link (dblink) provided in the command line to access tables on the primary database. The link must reference a privileged account that can read and lock the table on the primary database. |

See Section 4.2.17 for additional information about this SQL statement.

## 13.16 ALTER DATABASE {STOP | ABORT} LOGICAL STANDBY APPLY

This statement is for logical standby databases only.

This clause stops log apply services on a logical standby database. The SQL statement syntax is:

```
ALTER DATABASE { STOP | ABORT } LOGICAL STANDBY APPLY;
```

Table 13–14 describes the keywords for this statement.

*Table 13–14 Keywords for the {STOP | ABORT} LOGICAL STANDBY APPLY Clause*

| Keyword | Description |
|---|---|
| `STOP` | Stops log apply services in an orderly fashion so that you can make changes to logical standby settings or perform planned maintenance. This clause is useful for refreshing materialized views or function-based indexes. Log transport services will continue to send archived redo logs to the logical standby database. |

***Table 13–14 Keywords for the {STOP | ABORT} LOGICAL STANDBY APPLY Clause***

| Keyword | Description |
|---|---|
| ABORT | Stops log apply services immediately. If the DBA_LOGSTDBY_PARAMETERS view shows the TRANSACTION_CONSISTENCY option is set to READ_ONLY or NONE, an ABORT might leave transactions in an inconsistent manner. Only use the ABORT keyword when an immediate shutdown is necessary. |

# 14

# Views

This chapter describes the views that are used in a Data Guard environment. This is a subset of the views that are available for use in a database. This chapter contains the following sections:

- V$LOGFILE
- V$LOG_HISTORY
- V$LOGSTDBY (Logical Standby Databases Only)
- V$LOGSTDBY_STATS (Logical Standby Databases Only)
- V$MANAGED_STANDBY (Physical Standby Databases Only)
- V$STANDBY_LOG

## About Views

An Oracle database contains a set of underlying views that are maintained by the server and accessible to the database administrator. These **fixed views** are also called **dynamic performance views** because they are continuously updated while a database is open and in use, and their contents relate primarily to performance.

Although these views appear to be regular database tables, they are not. These views provide data on internal disk structures and memory structures. You can select from these views, but you can never update or alter them.

Fixed view names are usually prefixed with either V$, or GV$, for example, `V$ARCHIVE_DEST` or `GV$ARCHIVE_DEST`. The views that are prefixed with DBA_ display all relevant information in the entire database. Standard dynamic performance views (V$ fixed views) store information on the local instance. In contrast, global dynamic performance views (GV$ fixed views) store information on all open instances. Each V$ fixed view has a corresponding GV$ fixed view. DBA_ views are intended only for administrators. They can be accessed only by users with the `SELECT_ANY_TABLE` privilege. (This privilege is assigned to the DBA role when the system is initially installed.)

In most cases, the information available in fixed views persists across instance shutdowns. However, certain fixed view information is reset when the instance is shut down; these views are specifically identified in this chapter.

For additional information about views, see *Oracle9i Database Reference.*

# DBA_LOGSTDBY_EVENTS (Logical Standby Databases Only)

The DBA_LOGSTDBY_EVENTS view contains information about the activity of the logical standby database system. It can be used to determine the cause of failures that occur when log apply services apply redo logs. This view contains the following columns:

| Column | Datatype | Description |
|---|---|---|
| EVENT_TIME | DATE | Time the event was logged. |
| CURRENT_SCN | NUMBER | Change vector SCN for the event. If a failure occurred, examine this column to determine which archived redo log contains the source of the failure (for example, an unsupported record). |
| COMMIT_SCN | NUMBER | SCN value for which the change was committed. |
| XIDUSN | NUMBER | Transaction ID undo segment number. |
| XIDSLT | NUMBER | Transaction ID slot number. |
| XIDSQN | NUMBER | Transaction ID sequence number. |
| EVENT | CLOB | The statement that was being processed when the failure occurred. |
| STATUS_CODE | NUMBER | Status (or Oracle error code) belonging to the STATUS message. |
| STATUS | VARCHAR2(2000) | Description of the current activity of the process or the reason why the apply operation stopped. |

# DBA_LOGSTDBY_LOG (Logical Standby Databases Only)

The DBA_LOGSTDBY_LOG view shows the logs registered for a logical standby database. The view contains the following columns:

| Column | Datatype | Description |
|---|---|---|
| THREAD# | NUMBER | Thread ID of the archived redo log. The THREAD number is 1 for a single instance. For Real Application Clusters, this column will contain different numbers. |
| SEQUENCE# | NUMBER | Sequence number of the archived redo log. |
| FIRST_CHANGE# | NUMBER | SCN of the current archived redo log. |
| NEXT_CHANGE# | NUMBER | SCN of the next archived redo log. |
| FIRST_TIME | DATE | Date of the current archived redo log. |
| NEXT_TIME | DATE | Date of the next archived redo log. |
| FILE_NAME | VARCHAR2(3) | Name of the archived redo log. |
| TIMESTAMP | DATE | Time when the archived redo log was registered. |
| DICT_BEGIN | VARCHAR2(3) | Value Y or N, where Y indicates that the beginning of the dictionary build is in this particular archived redo log. |
| DICT_END | VARCHAR2(3) | Value Y or N, where Y indicates that the end of the dictionary build is in this particular archived redo log. |

**Note:** The SCN values in this view correlate to the SCN values in the DBA_LOGSTDBY_PROGRESS (Logical Standby Databases Only) view.

# DBA_LOGSTDBY_NOT_UNIQUE (Logical Standby Databases Only)

The DBA_LOGSTDBY_NOT_UNIQUE view identifies tables that have no primary and no non-null unique indexes. Most of the tables displayed in this view are supported because their columns contain enough information to be maintained in a logical standby database. Some tables, however, cannot be supported because their columns do not contain the necessary information. Unsupported tables usually contain a column defined using an unsupported datatype. This view contains the following columns:

| Column | Datatype | Description |
|---|---|---|
| OWNER | VARCHAR2(30) | Schema name |
| TABLE_NAME | VARCHAR2(30) | Name of the table |
| BAD_COLUMN | VARCHAR2(1) | This column contains a value of Y or N:<br>■ Y indicates the table column is defined using an unbounded datatype, such as LONG or BLOB. If two rows in the table match except in their LOB column, then the table cannot be maintained properly. Log apply services will attempt to maintain these tables, but you must ensure the application does not allow uniqueness only in the unbounded columns.<br>■ N indicates that enough column information is present to maintain the table in the logical standby database, but log transport services and log apply services would run more efficiently if you added a primary key. You should consider adding a disabled RELY constraint to these tables. |

# DBA_LOGSTDBY_PARAMETERS (Logical Standby Databases Only)

The DBA_LOGSTDBY_PARAMETERS view contains the list of parameters used by log apply services for logical standby databases. This view contains the following columns:

| Column | Datatype | Description |
|---|---|---|
| NAME | VARCHAR2 (30) | Name of the parameter. Possible values are:<br>■ MAX_SGA - System global area (SGA) allocated for the log apply services cache in megabytes.<br>■ MAX_SERVERS - Number of parallel query servers specifically reserved for log apply services.<br>■ MAX_EVENTS_RECORDED - Number of events stored in the DBA_LOGSTDBY_EVENTS view.<br>■ TRANSACTION_CONSISTENCY - Shows the level of transaction consistency maintained: FULL, READ_ONLY, or NONE.<br>■ RECORD_SKIP_ERRORS - Indicates records that are skipped.<br>■ RECORD_SKIP_DDL - Indicates skipped DDL statements.<br>■ RECORD_APPLIED_DDL - Indicates applied DDL statements.<br>■ FIRST_SCN - SCN at which log transport services will begin applying redo information.<br>■ PRIMARY - Database ID of the database to which logs are being applied.<br>■ LMNR_SID - LogMiner Session ID. This internal value indicates which LogMiner session is in use.<br>■ UNTIL_SCN - SCN value at which log apply services will shut down until all transactions are applied.<br>■ END_PRIMARY_SCN - During a switchover, this value indicates the last SCN applied by the new primary database from the old primary database.<br>■ NEW_PRIMARY_SCN - During a switchover, this value indicates the starting SCN for the new primary database.<br>■ COMPLETED_SESSION - Indicates that the log apply services session has concluded. The value will indicate SWITCHOVER or FAILOVER, as appropriate. |
| VALUE | VARCHAR2 (2000) | Value of the parameter |

# DBA_LOGSTDBY_PROGRESS (Logical Standby Databases Only)

The DBA_LOGSTDBY_PROGRESS view describes the progress of log apply services on the logical standby database. This view contains the following columns:

| Column | Datatype | Description |
|---|---|---|
| APPLIED_SCN | NUMBER | Shows the newest SCN at which all changes have been applied. The values in the APPLIED_SCN and NEWEST_SCN columns will match if all available redo log data was processed. |
| APPLIED_TIME | DATE | Estimate of the time and date of the APPLIED_SCN. |
| READ_SCN | NUMBER | All log data greater than this SCN was read and saved. |
| READ_TIME | DATE | Estimate of the time and date of the READ_SCN. |
| NEWEST_SCN | NUMBER | Most recent SCN available on the standby system. If no more logs are being transmitted to the standby database, changes could be applied to this SCN. The values in the APPLIED_SCN and NEWEST_SCN columns will match if all available redo log data has been processed. |
| NEWEST_TIME | DATE | Estimate of the time and date of the NEWEST_SCN. |

**Note:** The SCN values in this view correlate to the SCN values in the DBA_LOGSTDBY_LOG (Logical Standby Databases Only) view.

# DBA_LOGSTDBY_SKIP (Logical Standby Databases Only)

The DBA_LOGSTDBY_SKIP view lists the tables that will be skipped by log apply services. The DBA_LOGSTDBY_SKIP view contains the following columns:

| Column | Datatype | Description |
|---|---|---|
| ERROR | BOOLEAN | Indicates if the statement should be skipped or just returns errors for the statement. |
| STATEMENT_OPT | VARCHAR(30) | Specifies the type of statement that should be skipped. It must be one of the SYSTEM_AUDIT statement options. |
| SCHEMA | VARCHAR(30) | Name of the schema under which this skip option should be used. |
| NAME | VARCHAR(30) | Name of the option under which this skip option should be used. |
| PROC | VARCHAR(98) | Name of a stored procedure that will be executed when processing the skip option. |

# DBA_LOGSTDBY_SKIP_TRANSACTION (Logical Standby Databases Only)

The DBA_LOGSTDBY_SKIP_TRANSACTION view lists the skip settings chosen. This view contains the following columns:

| Column | Datatype | Description |
|---|---|---|
| XIDUSN | NUMBER | Transaction ID undo segment number |
| XIDSLT | NUMBER | Transaction ID slot number |
| XIDSQN | NUMBER | Transaction ID sequence number |

# DBA_LOGSTDBY_UNSUPPORTED (Logical Standby Databases Only)

The DBA_LOGSTDBY_UNSUPPORTED view identifies the schemas and tables (and columns in those tables) that contain unsupported datatypes. Use this view when you are preparing to create a logical standby database. This view contains the following columns:

| Column | Datatype | Description |
|---|---|---|
| OWNER | VARCHAR2(30) | Schema name of the unsupported table |
| TABLE_NAME | VARCHAR2(30) | Name of the unsupported table |
| COLUMN_NAME | VARCHAR2(30) | Name of the unsupported column |
| DATA_TYPE | VARCHAR2(106) | Datatype of the unsupported column |

# V$ARCHIVE_DEST

The V$ARCHIVE_DEST view describes, for the current instance, all the archived redo log destinations, their current value, mode, and status.

**Note:** The information in this view does not persist across an instance shutdown.

The V$ARCHIVE_DEST view contains the following columns:

| Column | Description |
|---|---|
| DEST_ID | Identifies the log archive destination parameter. |
| STATUS | Identifies the current status of the destination. Possible values are:<br>■ VALID - Initialized and available<br>■ INACTIVE - No destination information<br>■ DEFERRED - Manually disabled by the user<br>■ ERROR - Error during open or copy<br>■ DISABLED - Disabled after error<br>■ BAD PARAM - Error with the LOG_ARCHIVE_DEST_*n* parameter<br>■ ALTERNATE - Destination in an alternate state<br>■ FULL - Exceeded quota size for the destination |
| BINDING | Specifies how failure will affect the archival operation. Possible values are:<br>■ OPTIONAL - Successful archival operation not required<br>■ MANDATORY - Successful archival operation required |
| NAME_SPACE | Identifies the scope of parameter setting. Possible values are:<br>■ SYSTEM - System definition<br>■ SESSION - Session definition |
| TARGET | Specifies if the archive destination is local or remote to the primary database. Possible values are:<br>■ PRIMARY - Local<br>■ STANDBY - Remote |

| Column | Description |
|---|---|
| ARCHIVER | Identifies the archiver process relative to the database where the query is issued. Possible values are:<br>■ ARC*n*<br>■ FOREGROUND<br>■ LGWR<br>■ RFS |
| SCHEDULE | Indicates if the archiving of this destination is INACTIVE, PENDING, ACTIVE, or LATENT. |
| DESTINATION | Displays the location where the archived redo logs are to be archived. |
| LOG_SEQUENCE | Identifies the sequence number of the last archived redo log to be archived. |
| REOPEN_SECS | Identifies the retry time, in seconds, after an error. |
| DELAY_MINS | Identifies the delay interval, in minutes, before the archived redo log is automatically applied to a standby database. |
| PROCESS | Identifies the archiver process relative to the primary database, even if the query is issued on the standby database. Possible values are:<br>■ ARC*n*<br>■ FOREGROUND<br>■ LGWR |
| REGISTER | Indicates whether or not the archived redo log is registered in the remote destination control file. If the archived redo log is registered, it is available to the managed recovery operation. Possible values are:<br>■ YES<br>■ NO |
| FAIL_DATE | Indicates the date and time of the error. |
| FAIL_SEQUENCE | Indicates the sequence number of the archived redo log being archived when the last error occurred. |
| FAIL_BLOCK | Indicates the block number of the archived redo log being archived when the last error occurred. |
| FAILURE_COUNT | Identifies the current number of consecutive archival operation failures that occurred for the destination. |
| MAX_FAILURE | Allows you to control the number of times log transport services will attempt to reestablish communication and resume archival operations with a failed destination. |
| ERROR | Displays the error text. |

| Column | Description |
|---|---|
| ALTERNATE | Identifies the alternate destination, if any. |
| DEPENDENCY | Identifies the dependent archive destination, if any. |
| REMOTE_TEMPLATE | Displays the template to be used to derive the location to be recorded. |
| QUOTA_SIZE | Identifies the destination quotas, expressed in bytes. |
| QUOTA_USED | Identifies the size of all archived redo logs currently residing on the specified destination. |
| MOUNTID | Identifies the instance mount identifier. |
| AFFIRM | Displays the disk I/O mode. |
| ASYNC_BLOCKS | Specifies the number of blocks for the ASYNC attribute. |
| TRANSMIT_MODE | Displays network transmission mode. Possible values are:<br>■ PARALLELSYNC<br>■ SYNCHRONOUS<br>■ ASYNCHRONOUS |
| TYPE | Indicates if the archived log destination definition is PUBLIC or PRIVATE. Only PUBLIC destinations can be modified at runtime using the ALTER SYSTEM SET or ALTER SESSION SET statements. By default, all archived log destinations are PUBLIC. |
| NET_TIMEOUT | Specifies the number of seconds the log writer process will wait for status from the network server of a network operation issued by the log writer process. |

# V$ARCHIVE_DEST_STATUS

The V$ARCHIVE_DEST_STATUS view displays runtime and configuration information for the archived redo log destinations.

**Note:** The information in this view does not persist across an instance shutdown.

The V$ARCHIVE_DEST_STATUS view contains the following columns:

| Column | Description |
|---|---|
| DEST_ID | Identifies the log archive destination parameter. |
| STATUS | Identifies the current status of the destination. Possible values are:<br>■ VALID - Initialized and available<br>■ INACTIVE - No destination information<br>■ DEFERRED - Manually disabled by the user<br>■ ERROR - Error during open or copy<br>■ DISABLED - Disabled after error<br>■ BAD PARAM - Error with the LOG_ARCHIVE_DEST_*n* parameter<br>■ ALTERNATE - Destination in an alternate state<br>■ FULL - Exceeded quota size for the destination |
| TYPE | Identifies the type of archival destination database. Possible values are:<br>■ LOCAL - Local to primary instance<br>■ PHYSICAL - Physical standby database<br>■ CROSS-INSTANCE - An instance of the primary database |
| DATABASE_MODE | Identifies the current mode of the archival destination database. Possible values are:<br>■ STARTED - Instance started, not mounted<br>■ MOUNTED - Mounted<br>■ MOUNTED-STANDBY - Mounted standby<br>■ OPEN - Open read / write<br>■ OPEN_READ-ONLY - Open read-only |

| Column | Description |
|---|---|
| RECOVERY_MODE | Identifies the current mode of media recovery at the archival destination database. Possible values are:<br>■ IDLE - Managed recovery not active<br>■ MANUAL - Manual media recovery active<br>■ MANAGED - Managed recovery active |
| DESTINATION | Displays the location where the archived redo logs are to be archived. |
| ARCHIVED_THREAD# | Identifies the thread number of the most recent archived redo log received at the destination. |
| ARCHIVED_SEQ# | Identifies the log sequence number of the most recent archived redo log received at the destination. |
| APPLIED_THREAD# | Identifies the thread number of the most recent applied redo log received at the destination. |
| APPLIED_SEQ# | Identifies the log sequence number of the most recent applied redo log received at the destination. |
| ERROR | Displays the error text. |
| STANDBY_LOGFILE_COUNT | Indicates the total number of standby redo logs created on the standby database. |
| STANDBY_LOGFILE_ACTIVE | Indicates the total number of standby redo logs on the standby database that are active and contain primary database online redo log information |
| PROTECTION_MODE | Indicates if and how the database is protected. Possible values are:<br>■ MAXIMUM PROTECTION<br>■ MAXIMUM AVAILABILITY<br>■ RESYNCHRONIZATION<br>■ MAXIMUM PERFORMANCE<br>■ UNPROTECTED |
| SRL | Indicates the use of standby redo logs on the standby database. Possible values are:<br>■ YES<br>■ NO |

# V$ARCHIVE_GAP

The V$ARCHIVE_GAP view displays information to help you identify an archive gap. The V$ARCHIVE_GAP view contains the following columns:

| Column | Description |
|---|---|
| THREAD# | Specifies the thread number |
| LOW_SEQUENCE# | Specifies the low number of the log |
| HIGH_SEQUENCE# | Specifies the high number of the log |

# V$ARCHIVED_LOG

The V$ARCHIVED_LOG view displays archived redo log information from the control file, including archived log names. This view contains the following columns:

| Column | Description |
|---|---|
| RECID | Archived log record ID. |
| STAMP | Archived log record stamp. |
| NAME | Archived log filename. If set to NULL, the log was cleared before it was archived. |
| DEST_ID | The original destination from which the archived log was generated. Value is 0 if the destination identifier is not available. |
| THREAD# | Redo thread number. |
| SEQUENCE# | Redo log sequence number. |
| RESETLOGS_CHANGE# | Resetlogs change number of the database when this log was written. |
| RESETLOGS_TIME | Resetlogs time of the database when this log was written. |
| FIRST_CHANGE# | First change number in the archived log. |
| FIRST_TIME | Timestamp of the first change. |
| NEXT_CHANGE# | First change in the next log. |
| NEXT_TIME | Timestamp of the next change. |
| BLOCKS | Size of the archived log in blocks. |
| BLOCK_SIZE | Redo log block size. This is the logical block size of the archived log, which is the same as the logical block size of the online log from which this archived log was copied. The online log logical block size is a platform-specific value that is not adjustable by the user. |
| CREATOR | Identifies the creator of the archived log. |
| REGISTRAR | Identifies the registrar of the entry. |
| STANDBY_DEST | Indicates if the entry is an archived log destination. |

| Column | Description |
|---|---|
| ARCHIVED | Indicates that the online redo log was archived or that RMAN only inspected the log and created a record for future application of redo logs during recovery. |
| APPLIED | Indicates whether or not the archived log was applied to its corresponding standby database. |
| DELETED | Specifies whether or not an RMAN DELETE command has physically deleted the archived redo log from disk, as well as logically removing it from the control file of the **target database** and from the **recovery catalog**. |
| STATUS | The status of this archived log. Possible values are:<br>■ A - Available<br>■ D - Deleted<br>■ U - Unavailable<br>■ X - Expired |
| COMPLETION_TIME | Indicates the time when the archiving completed. |
| DICTIONARY_BEGIN | Indicates whether or not this log contains the start of a LogMiner dictionary. |
| DICTIONARY_END | Indicates whether or not this log contains the end of a LogMiner dictionary. |
| BACKUP_COUNT | Indicates the number of times this file was backed up. Values range from 0 to 15. If the file was backed up more than 15 times, the value remains 15. |
| END_OF_REDO | Indicates whether or not this archived redo log contains the end of all redo information from the primary database. Values are YES and NO. |
| ARCHIVAL_THREAD# | Indicates the redo thread number of the instance that performed the archival operation. This column differs from the THREAD# column only when a closed thread is archived by another instance. |
| ACTIVATION# | Indicates the number assigned to the database instantiation. |

# V$DATABASE

The V$DATABASE view provides database information from the control file. This view contains the following columns:

| Column | Description |
|---|---|
| DBID | The database identifier that is calculated when the database is created. This identifier is stored in all file headers. |
| NAME | Name of the database. |
| CREATED | Creation date. |
| RESETLOGS_CHANGE# | Change number at open resetlogs. |
| RESETLOGS_TIME | Timestamp of open resetlogs. |
| PRIOR_RESETLOGS_ CHANGE# | Change number at prior resetlogs. |
| PRIOR_RESETLOGS_ TIME | Timestamp of prior resetlogs. |
| LOG_MODE | Archive log mode. |
| CHECKPOINT_CHANGE# | Last SCN checkpointed. |
| ARCHIVE_CHANGE# | Last SCN archived. |
| CONTROLFILE_TYPE | The type of control file. Possible values are:<br>■ STANDBY - Indicates the database is in standby mode.<br>■ LOGICAL - Indicates the database is a logical standby database.<br>■ CLONE - Indicates a clone database.<br>■ BACKUP \| CREATED - Indicates the database is being recovered using a backup or created control file.<br>■ CURRENT - Indicates the database is available for general use. |
| CONTROLFILE_CREATED | Control file creation timestamp. |
| CONTROLFILE_ SEQUENCE# | Control file sequence number incremented by control file transactions. |
| CONTROLFILE_CHANGE# | Last change number in the backup control file. Set to NULL if the control file is not a backup. |

| Column | Description |
|---|---|
| CONTROLFILE_TIME | Last timestamp in the backup control file. Set to NULL if the control file is not a backup. |
| OPEN_RESETLOGS | Indicates if the next database open allows or requires the resetlogs option. |
| VERSION_TIME | The version time. |
| OPEN_MODE | Open mode information. |
| PROTECTION_MODE | Indicates if and how the database is protected. Possible values are:<br>■ MAXIMUM PROTECTION<br>■ MAXIMUM AVAILABILITY<br>■ RESYNCHRONIZATION<br>■ MAXIMUM PERFORMANCE<br>■ UNPROTECTED |
| PROTECTION_LEVEL | Displays the aggregated protection mode currently in effect on the primary or standby database. Possible values are:<br>■ MAXIMUM PROTECTION<br>■ MAXIMUM AVAILABILITY<br>■ RESYNCHRONIZATION<br>■ MAXIMUM PERFORMANCE<br>■ UNPROTECTED |
| REMOTE_ARCHIVE | The value of the REMOTE_ARCHIVE_ENABLE initialization parameter. Possible values are:<br>■ TRUE<br>■ FALSE<br>■ SEND<br>■ RECEIVE |
| ACTIVATION# | Number assigned to the database instantiation. |
| DATABASE_ROLE | Current role of the database; either primary or standby. |
| ARCHIVELOG_CHANGE# | Highest NEXT_CHANGE# (from the V$ARCHIVED_LOG view) for an archived log. |

| Column | Description |
|---|---|
| SWITCHOVER_STATUS (Physical Standby Databases Only) | Specifies if switchover is allowed. This column currently is supported only for use with physical standby databases. Possible values are:<br>■ NOT ALLOWED - Either this is a standby database and the primary database has not been switched first, or this is a primary database and there are no standby databases.<br>■ SESSIONS ACTIVE - Indicates that there are active SQL sessions attached to the primary or standby database that need to be disconnected before the switchover operation is permitted. Query the V$SESSION view to identify the specific processes that need to be terminated.<br>■ SWITCHOVER PENDING - This is a standby database and the primary database switchover request has been received but not processed.<br>■ SWITCHOVER LATENT - The switchover was in pending mode, but did not complete and went back to the primary database.<br>■ TO PRIMARY - This is a standby database and is allowed to switch over to a primary database.<br>■ TO STANDBY - This is a primary database and is allowed to switch over to a standby database.<br>■ RECOVERY NEEDED - This is a standby database that has not received the switchover request. |
| GUARD_STATUS | Protects data from being changed. Possible values are:<br>■ ALL - Indicates all users other than SYS are prevented from making changes to any data in the database.<br>■ STANDBY - Indicates all users other than SYS are prevented from making changes to any database object being maintained by logical standby.<br>■ NONE - Indicates normal security for all data in the database. |
| SUPPLEMENTAL_LOG_ DATA_MIN | Ensures that LogMiner will have sufficient information to support chained rows and various storage arrangements such as cluster tables.<br>See *Oracle9i SQL Reference* for additional information about the ALTER DATABASE ADD SUPPLEMENTAL LOG DATA statement. |
| SUPPLEMENTAL_LOG_ DATA_PK | For all tables with a primary key, ensures that all columns of the primary key are placed into the redo log whenever an update operation is performed.<br>See *Oracle9i SQL Reference* for additional information about the ALTER DATABASE ADD SUPPLEMENTAL LOG DATA statement. |
| SUPPLEMENTAL_LOG_ DATA_UI | For all tables with a unique key, ensures that if any unique key columns are modified, all other columns belonging to the unique key are also placed into the redo log.<br>See *Oracle9i SQL Reference* for additional information about the ALTER DATABASE ADD SUPPLEMENTAL LOG DATA statement. |

| Column | Description |
|---|---|
| FORCE_LOGGING | Redo generation is forced even for NOLOGGING operations. Possible values are:<br>■ YES<br>■ NO |
| DATAGUARD_BROKER | Indicates if the Data Guard configuration is being managed by the broker. Possible values are:<br>■ ENABLED indicates the configuration is under the control of the broker<br>■ DISABLED indicates the configuration is not under the control of the broker |

# V$DATAFILE

The V$DATAFILE view provides datafile information from the control file. This view contains the following columns:

| Column | Description |
|---|---|
| FILE# | File identification number. |
| CREATION_CHANGE# | Change number at which the datafile was created. |
| CREATION_TIME | Timestamp of the datafile creation. |
| TS# | Tablespace number. |
| RFILE# | Tablespace relative datafile number. |
| STATUS | Type of file (system or user) and its status. Possible values are:<br>■ OFFLINE - cannot be written to<br>■ ONLINE - can be written to<br>■ SYSTEM - system datafile<br>■ RECOVER - needs recovery<br>■ SYSOFF - offline system |
| ENABLED | Describes how accessible the file is from SQL. Possible values are:<br>■ DISABLED - no SQL access allowed<br>■ READ ONLY - no SQL updates allowed<br>■ READ WRITE - full access allowed |
| CHECKPOINT_CHANGE# | SCN at last checkpoint. |
| CHECKPOINT_TIME | Timestamp of the last checkpoint. |
| UNRECOVERABLE_ CHANGE# | Last unrecoverable change number made to this datafile. This column is always updated when an unrecoverable operation completes. |
| UNRECOVERABLE_TIME | Timestamp of the last unrecoverable change. |
| LAST_CHANGE# | Last change number made to this datafile. Set to NULL if the datafile is being changed. |
| LAST_TIME | Timestamp of the last change. |
| OFFLINE_CHANGE# | Offline change number of the last offline range. This column is updated only when the datafile is brought online. |

| Column | Description |
|---|---|
| ONLINE_CHANGE# | Online change number of the last offline range. |
| ONLINE_TIME | Online timestamp of the last offline range. |
| BYTES | Current datafile size in bytes; 0 if inaccessible. |
| BLOCKS | Current datafile size in blocks; 0 if inaccessible. |
| CREATE_BYTES | Size when created, in bytes. |
| BLOCK_SIZE | Block size of the datafile. |
| NAME | Datafile name. |
| PLUGGED_IN | Describes if the tablespace is plugged in. The value is 1 if the tablespace is plugged in and has not been made read / write; 0 if not. |
| BLOCK1_OFFSET | The offset from the beginning of the file to where the Oracle generic information begins. The exact length of the file can be computed as follows: BYTES + BLOCK1_OFFSET. |
| AUX_NAME | The auxiliary name that has been set for this file. |

# V$DATAGUARD_STATUS

The V$DATAGUARD_STATUS view displays and logs events that would typically be triggered by any message to the alert log or server process trace files.

**Note:** The information in this view does not persist across an instance shutdown.

The V$DATAGUARD_STATUS view contains the following columns:

| Column | Description |
|---|---|
| INST_ID | The ID of the instance encountering the event. This column is present in the GV$DATAGUARD_STATUS view and not in the V$DATAGUARD_STATUS view. |
| FACILITY | Facility that encountered the event. Possible values are:<br>■ CRASH RECOVERY<br>■ LOG TRANSPORT SERVICES<br>■ LOG APPLY SERVICES<br>■ ROLE MANAGEMENT SERVICES<br>■ REMOTE FILE SERVER<br>■ FETCH ARCHIVE LOG<br>■ DATA GUARD<br>■ NETWORK SERVICES |
| SEVERITY | The severity of the event. Possible values are:<br>■ INFORMATIONAL - informational message<br>■ WARNING - warning message<br>■ ERROR - indicates the process has failed<br>■ FATAL-indicates the process, the database, or both have failed<br>■ CONTROL - an expected change in state, such as the start or completion of an archival, log recovery, or switchover operation |
| DEST_ID | The destination ID number to which the event pertains. If the event does not pertain to a particular destination, the value is 0. |
| MESSAGE_NUM | A chronologically increasing number giving each event a unique number. |

| Column | Description |
|---|---|
| ERROR_CODE | The error ID pertaining to the event. |
| CALLOUT | Indicates whether or not the current entry is a callout event. Possible values are:<br>■ YES<br>■ NO<br>A YES value indicates that this event might require the DBA to perform some action. Examine the ERROR_CODE and MESSAGE columns for more information.<br>A NO value generally corresponds to an INFORMATIONAL or WARNING event that does not require any action by the DBA. |
| TIMESTAMP | The date and time when the entry was created. |
| MESSAGE | A text message describing the event. |

# V$LOG

The V$LOG view contains log file information from the online redo logs. This view contains the following columns:

| Column | Description |
|---|---|
| GROUP# | Log group number. |
| THREAD# | Log thread number. |
| SEQUENCE# | Log sequence number. |
| BYTES | Size of the log in bytes. |
| MEMBERS | Number of members in the log group. |
| ARCHIVED | Archive status. |
| STATUS | Indicates the log status. Possible values are:<br>■ UNUSED - The online redo log has never been written to. This is the status of a redo log that was just added, or just after specifying a RESETLOGS option when it is not the current redo log.<br>■ CURRENT - This is the current redo log. This implies that the redo log is active. The redo log could be open or closed.<br>■ ACTIVE - The log is active but is not the current log. It is needed for failure recovery. It might be in use for block recovery. It might or might not be archived.<br>■ CLEARING - The log is being re-created as an empty log after an ALTER DATABASE CLEAR LOGFILE statement. After the log is cleared, the status changes to UNUSED.<br>■ CLEARING_CURRENT - The current log is being cleared of a closed thread. The log can stay in this status if there is some failure in the switch, such as an I/O error writing the new log header.<br>■ INACTIVE - The log is no longer needed for instance recovery. It might be in use for managed recovery. It might or might not be archived.<br>■ INVALIDATED - Archived the current redo log without a log switch |
| FIRST_CHANGE# | Lowest SCN in the log. |
| FIRST_TIME | Time of first SCN in the log. |

# V$LOGFILE

The V$LOGFILE view contains information about the online redo logs. This view contains the following columns:

| Column | Description |
|---|---|
| GROUP# | Redo log group identifier number. |
| STATUS | Status of this log member. Possible values are:<br>■ INVALID - File is inaccessible.<br>■ STALE - Contents are incomplete.<br>■ DELETED - File is no longer used.<br>■ blank (no value listed) - File is in use. |
| MEMBER | Redo log member name |
| TYPE | Specifies if this is a standby log or an online log. Possible values are:<br>■ STANDBY<br>■ ONLINE |

# V$LOG_HISTORY

The V$LOG_HISTORY view contains log history information from the control file. This view contains the following columns:

| Column | Description |
|---|---|
| RECID | Control file record ID |
| STAMP | Control file record stamp |
| THREAD# | Thread number of the archived log |
| SEQUENCE# | Sequence number of the archived log |
| FIRST_CHANGE# | Lowest SCN in the log |
| FIRST_TIME | Time of first entry (lowest SCN) in the log |
| NEXT_CHANGE# | Highest SCN in the log |

# V$LOGSTDBY (Logical Standby Databases Only)

The V$LOGSTDBY view provides dynamic information about what is happening to log apply services. This view is very helpful when you are diagnosing performance problems during the logical application of archived redo logs to the standby database, and it can be helpful for other problems. The V$LOGSTDBY view contains the following columns:

| Column | Datatype | Description |
|---|---|---|
| SERIAL# | NUMBER | Contains the SQL session serial number. This data is used when joining this view with V$SESSION and V$PX_SESSION views. |
| LOGSTDBY_ID | NUMBER | Contains the parallel query slave ID. |
| PID | VARCHAR2(9) | Contains the process ID. |
| TYPE | VARCHAR2(30) | Indicates the task being performed by the process: COORDINATOR, APPLIER, ANALYZER, READER, PREPARER, BUILDER. |
| STATUS_CODE | NUMBER | Contains the status number (or Oracle error code) belonging to the STATUS message. |
| STATUS | VARCHAR2(256) | Description of the current activity of the process. |
| HIGH_SCN | NUMBER | Contains the highest SCN seen by the process. This column is used to confirm the progress of the individual process. |

# V$LOGSTDBY_STATS (Logical Standby Databases Only)

The V$LOGSTDBY_STATS view displays LogMiner statistics, current state, and status information for a logical standby database during SQL apply operations. If log apply services are not running, the values for the statistics are cleared. This view contains the following columns:

| Column | Datatype | Description |
|---|---|---|
| NAME | VARCHAR2(64) | Name of the statistic, state, or status:<br>**Note:** Many of the following statistics are subject to change or deletion; programmers should write application code to tolerate missing or extra statistics.<br>■ Number of preparers<br>■ Number of appliers<br>■ Maximum SGA for LCR cache<br>■ Parallel servers in use<br>■ Transaction consistency<br>■ Coodinator state<br>■ Transactions scheduled<br>■ Transactions applied<br>■ Preparer memory allocation failures<br>■ Builder memory allocation failures<br>■ Attempts to handle low memory<br>■ Successful low memory recovery<br>■ Memory spills avoided<br>■ Rollback attempts<br>■ Successful rollbacks<br>■ Memory spill attempts<br>■ Successful memory spills<br>■ Preparer ignored memory low water mark<br>■ Builder ignored memory low water mark<br>■ Mining resumed |
| VALUE | VARCHAR2(64) | The value of the statistic or state information |

# V$MANAGED_STANDBY (Physical Standby Databases Only)

The V$MANAGED_STANDBY view displays current status information for Oracle database server processes related to physical standby databases in the Data Guard environment. The V$MANAGED_STANDBY view contains the columns shown in the following table; the information does not persist after an instance shutdown.

| Column | Description |
|---|---|
| PROCESS | Type of process whose information is being reported. Possible values are:<br>■ ARCH - archiver process<br>■ RFS - remote file server<br>■ MRP0 - detached recovery server process<br>■ MR(fg) - foreground recovery session |
| PID | Operating system identifier of the process. |
| STATUS | Current process status. Possible values are:<br>■ UNUSED - No active process.<br>■ ALLOCATED - Process is active but not currently connected to a primary database.<br>■ CONNECTED - Network connection is established to a primary database.<br>■ ATTACHED - Process is attached to, and communicating with, a primary database.<br>■ IDLE - Process is not performing any activities.<br>■ ERROR - Process has failed.<br>■ OPENING - Process is opening the archived redo log.<br>■ CLOSING - Process has completed the archival operation and is closing the archived redo log.<br>■ WRITING - Process is actively writing archived redo log data.<br>■ RECEIVING - Process is receiving network communication.<br>■ ANNOUNCING - Process is announcing the existence of a potential dependent archived redo log.<br>■ REGISTERING - Process is registering the existence of a completed dependent archived redo log.<br>■ WAIT_FOR_LOG - Process is waiting for the archived redo log to be completed.<br>■ WAIT_FOR_GAP - Process is waiting for the archive gap to be resolved.<br>■ APPLYING_LOG - Process is applying the archived redo log to the standby database. |

| Column | Description |
|---|---|
| CLIENT_PROCESS | Identifies the corresponding primary database process. Possible values are:<br>■ ARCHIVAL - foreground (manual) archival process (SQL)<br>■ ARCH - background ARC*n* process<br>■ LGWR - background LGWR process |
| CLIENT_PID | Operating system identifier of the client process. |
| CLIENT_DBID | Database identifier of the primary database. |
| GROUP# | Standby redo log group. |
| THREAD# | Archived redo log thread number. |
| SEQUENCE# | Archived redo log sequence number. |
| BLOCK# | Last processed archived redo log block number. |
| BLOCKS | Size of the archived redo log in 512-byte blocks. |
| DELAY_MINS | Archived redo log delay interval in minutes. |
| KNOWN_AGENTS | Total number of standby database agents processing an archived redo log. |
| ACTIVE_AGENTS | Number of standby database agents actively processing an archived redo log. |

# V$STANDBY_LOG

The V$STANDBY_LOG view contains the following columns:

| Column | Description |
|---|---|
| GROUP# | Log group number. |
| THREAD# | Log thread number. |
| SEQUENCE# | Log sequence number. |
| BYTES | Size of the log in bytes. |
| USED | Number of bytes used in the log. |
| ARCHIVED | Archive status. |
| STATUS | Indicates the log status. Possible values are:<br>■ UNUSED - The online redo log has never been written to. This is the status of a redo log that was just added, or just after specifying a RESETLOGS option when it is not the current redo log.<br>■ CURRENT - This is the current redo log. This implies that the redo log is active. The redo log could be open or closed.<br>■ ACTIVE - The log is active but is not the current log. It is needed for failure recovery. It might be in use for block recovery. It might or might not be archived.<br>■ CLEARING - The log is being re-created as an empty log after an ALTER DATABASE CLEAR LOGFILE statement. After the log is cleared, the status changes to UNUSED.<br>■ CLEARING_CURRENT - The current log is being cleared of a closed thread. The log can stay in this status if there is some failure in the switch, such as an I/O error writing the new log header.<br>■ INACTIVE - The log is no longer needed for instance recovery. It might be in use for managed recovery. It might or might not be archived.<br>■ INVALIDATED - Archived the current redo log without a log switch. |
| FIRST_CHANGE# | Lowest SCN in the log. |
| FIRST_TIME | Time of first SCN in the log. |
| LAST_CHANGE# | Last change number made to this datafile. Set to NULL if the datafile is being changed. |
| LAST_TIME | Timestamp of the last change. |

# Part III

## Appendixes and Glossary

This part contains the following:

# A

# Troubleshooting the Standby Database

This appendix provides help troubleshooting a standby database. This appendix contains the following sections:



## A.1 Problems During Standby Database Preparation

If you encounter a problem during standby database preparation, it will probably be one of the following:



### A.1.1 The Standby Archive Destination Is Not Defined Properly

If the `STANDBY_ARCHIVE_DEST` initialization parameter is not defined as a valid directory name on the standby site, the Oracle database server will not be able to determine the directory in which to store the archived redo logs. Check the

DESTINATION and ERROR columns in the V$ARCHIVE_DEST view. For example, enter:

```
SQL> SELECT DESTINATION, ERROR FROM V$ARCHIVE_DEST;
```

Make sure the destination is valid.

## A.1.2 The Standby Site Does Not Receive Logs Archived by the Primary Database

If the standby site is not receiving the logs, the first thing you should do is obtain information about the archiving status of the primary database by querying the V$ARCHIVE_DEST view. Check especially for error messages. For example, enter the following query:

```
SQL> SELECT DEST_ID "ID",
  2> STATUS "DB_status",
  3> DESTINATION "Archive_dest",
  4> ERROR "Error"
  5> FROM V$ARCHIVE_DEST;

ID DB_status Archive_dest                  Error
-- --------- ------------------------------ ----------------------------------
 1  VALID    /vobs/oracle/work/arc_dest/arc
 2  ERROR    standby1                       ORA-16012: Archivelog standby database identifier mismatch
 3  INACTIVE
 4  INACTIVE
 5  INACTIVE
5 rows selected.
```

If the output of the query does not help you, check the following list of possible issues. If any of the following conditions exist, the primary database will fail to archive to the standby site:

- The service name for the standby instance is not configured correctly in the tnsnames.ora file at the primary site.
- The service name listed in the LOG_ARCHIVE_DEST_*n* parameter of the primary initialization parameter file is incorrect.
- The LOG_ARCHIVE_DEST_STATE_*n* parameter specifying the state of the standby archiving destination has the value DEFER.
- The listener.ora file has not been configured correctly at the standby site.
- The listener is not started.
- The standby instance is not started.

- You have added a standby archiving destination to the primary initialization parameter file, but have not yet enabled the change.
- You used an invalid backup as the basis for the standby database (for example, you used a backup from the wrong database, or did not create the standby control file using the correct method).

### A.1.3 You Cannot Mount the Physical Standby Database

If any of the following conditions exist, you cannot mount the physical standby database:

- The standby instance is not started in NOMOUNT mode. You must first start the instance and *then* mount the database.
- The standby control file was not created with the ALTER DATABASE CREATE STANDBY CONTROLFILE ... statement or RMAN. 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 STANDBY option

## A.2 Log Destination Failures

If you specify REOPEN for an OPTIONAL destination, it is possible for the Oracle database server to reuse online redo logs even if there is an error. If you specify REOPEN for a MANDATORY destination, the log transport services component stalls the primary database when it cannot successfully archive redo logs.

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'
```

Using the ALTERNATE attribute of the LOG_ARCHIVE_DEST_*n* parameter, you can specify alternate archive destinations. An alternate archive destination can be used when the archiving of an online redo log to a standby site fails. If archiving fails and the NOREOPEN attribute has been specified, or the MAX_FAILURE attribute threshold has been exceeded, log transport services will attempt to archive redo logs to the alternate destination on the next archiving 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 parameter file so that a single, mandatory, local destination will automatically fail over to a different destination if any error occurs.

***Example A–2 Specifying an Alternate Destination***

```
LOG_ARCHIVE_DEST_1='LOCATION=/disk1 MANDATORY ALTERNATE=LOG_ARCHIVE_DEST_2'
LOG_ARCHIVE_DEST_STATE_1=ENABLE
LOG_ARCHIVE_DEST_2='LOCATION=/disk2 MANDATORY'
LOG_ARCHIVE_DEST_STATE_2=ENABLE
LOG_ARCHIVE_DEST_STATE_2=ALTERNATE
```

If the LOG_ARCHIVE_DEST_1 destination fails, the archiving process will automatically switch to the LOG_ARCHIVE_DEST_2 destination at the next log switch on the primary database.

## A.3 Ignoring Logical Standby Database Failures

An important skip tool is DBMS_LOGSTDBY.SKIP_ERROR. 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 that runtime determinations can be made whether to skip the statement, execute this statement, or execute a replacement statement

> **See Also:** *Oracle9i Supplied PL/SQL Packages and Types Reference* for more information about using the DBMS_LOGSTDBY package with PL/SQL callout procedures

Taking one of these actions prevents the SQL apply operations from stopping. Later, you can query the DBA_LOGSTDBY_EVENTS view to find and correct any problems that exist.

## A.4 Problems Switching Over to a Standby Database

If you encounter a problem switching over from a primary database to a standby database, it will probably be one of the following:

## A.4.1 Switchover Fails

`ALTER DATABASE COMMIT TO SWITCHOVER` failed with ORA-01093 error "Alter database close only permitted with no sessions connected."

This error occurs because the `COMMIT TO SWITCHOVER` statement implicitly closed the database and, if there are any other user sessions connected to the database, the close fails.

Action: Make sure all user sessions are disconnected from the database. You can query the `V$SESSION` fixed view to see what sessions are still around. For example:

```
SQL> SELECT SID, PROCESS, PROGRAM FROM V$SESSION;

       SID PROCESS   PROGRAM
---------- --------- -------------------------------------------------
         1 26900     oracle@dbuser-sun (PMON)
         2 26902     oracle@dbuser-sun (DBW0)
         3 26904     oracle@dbuser-sun (LGWR)
         4 26906     oracle@dbuser-sun (CKPT)
         5 26908     oracle@dbuser-sun (SMON)
         6 26910     oracle@dbuser-sun (RECO)
         7 26912     oracle@dbuser-sun (ARC0)
         8 26897     sqlplus@dbuser-sun (TNS V1-V3)
        11 26917     sqlplus@dbuser-sun (TNS V1-V3)

9 rows selected.
```

In the previous example, the first seven sessions are all server background processes. Among the two SQL*Plus sessions, one is the current SQL*Plus session issuing the query, and the other is an extra session that should be disconnected before the switchover operation.

## A.4.2 Recovering After An Unsuccessful Switchover Operation

In most cases, following the steps described in Section 7.2.1 will result in a successful switchover operation. However, if the switchover operation is initially unsuccessful, you might still be able to use one of the following recovery options to complete the switchover operation successfully.

### Option 1: Continue the current switchover operation.

If the switchover operation does not complete successfully, you can query the SEQUENCE# column in the V$ARCHIVED_LOG view to see if the last archived log was archived and applied on the old physical standby database. If the last log was not archived to the old physical standby database, you can manually copy the archived log from the old primary database to the old physical standby database and register it with the SQL ALTER DATABASE REGISTER LOGFILE filespec statement. If you then start up the managed recovery process, the archived log will be applied automatically. Query the SWITCHOVER_STATUS column in the V$DATABASE view. The TO PRIMARY value in the SWITCHOVER_STATUS column verifies that switchover to the primary role is now possible.

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

> **See Also:** Chapter 14 for information about other valid values for the SWITCHOVER_STATUS column of the V$DATABASE view

To continue with the switchover operation, return to Section 7.2.1 Step 5, and try again to switch the target physical standby database to the primary role.

### Option 2: Rollback the unsuccessful switchover operation and start over.

In situations where an error has occurred and it is not possible to continue with the switchover operation, it might still be possible to revert the new physical standby back to the primary role by using the following steps:

1. When the switchover operation to change the role from primary to standby was initiated, a trace file was written in the log directory. This trace file contains the SQL statements required to re-create the original primary control file. Locate the trace file and extract the SQL statements into a temporary file. Execute the temporary file from SQL*Plus. This will revert the new physical standby database back to the primary role.

2. Shut down the original physical standby database.
3. Create a new physical standby control file. This is necessary to resynchronize the primary database and physical standby database. Copy the standby control file to the original physical standby site.

   **See Also:** Section 3.2.3 for information about creating a standby control file.

4. Restart the original physical standby instance.

   If this procedure is successful and archive gap management is enabled, the FAL processes will start and re-archive any missing archived redo logs 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 that the archived redo log sequence numbers are correct.

   **See Also:** Section 6.4 for information about archive gap management and Section 6.7 for information about locating the trace files.

5. Try the switchover operation again.

   At this point, the Data Guard configuration has been rolled back to its initial state, and you can try the switchover operation again (after correcting any problems that might have led to the initial unsuccessful switchover operation).

### A.4.3 Startup of Second Physical Standby Database Fails

Suppose the standby database and the primary database reside on the same site. After both the `ALTER DATABASE COMMIT TO SWITCHOVER TO PHYSICAL STANDBY` and the `ALTER DATABASE COMMIT TO SWITCHOVER TO PRIMARY` statements are successfully executed, shut down and restart the physical standby database and the primary database. However, the startup of the second database fails with ORA-01102 error "cannot mount database in `EXCLUSIVE` mode."

This could happen during the switchover if you forget to set the `LOCK_NAME_SPACE` parameter in the initialization parameter file that is used by the standby database (that is, the original primary database). If the `LOCK_NAME_SPACE` 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 LOCK_NAME_SPACE=*unique_lock_name* to the initialization parameter file used by the physical standby database and shut down and restart both the standby and the primary databases.

## A.4.4 Archived Redo Logs Are Not Applied After a Switchover

The archived redo logs are not applied to the standby database after the switchover.

This might happen because some environment or initialization parameters have not been properly set after the switchover.

Action:

- Check the tnsnames.ora file at the primary site and the listener.ora file at the standby site. There should be entries for a listener at the standby site and a corresponding tnsname 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 has been set to properly archive logs from the primary site to 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 logs are applied to the desired location.
- At the standby site, set the DB_FILE_NAME_CONVERT and LOG_FILE_NAME_CONVERT initialization parameters. Set the STANDBY_FILE_MANAGEMENT initialization parameter to AUTO if you want the standby site to automatically add new datafiles that are created at the primary site.

## A.4.5 Switchover Fails When SQL Sessions Are Active

If you do not include the WITH SESSION SHUTDOWN clause as a part of the ALTER DATABASE COMMIT TO SWITCHOVER TO PHYSICAL STANDBY statement, active SQL sessions might prevent a switchover from being processed. Active SQL sessions can include other Oracle processes.

When sessions are active, an attempt to switch over fails with the following error message:

```
SQL> ALTER DATABASE COMMIT TO SWITCHOVER TO PHYSICAL STANDBY;
ALTER DATABASE COMMIT TO SWITCHOVER TO PHYSICAL STANDBY *
ORA-01093: ALTER DATABASE CLOSE only permitted with no sessions connected
```

Action: Query the V$SESSION view to determine which processes are causing the error. For example:

```
SQL> SELECT SID, PROCESS, PROGRAM FROM V$SESSION
  2> WHERE TYPE = 'USER'
  3>  AND SID <> (SELECT DISTINCT SID FROM V$MYSTAT);
SID        PROCESS   PROGRAM
---------  --------  ------------------------------------------------
        7      3537  oracle@nhclone2 (CJQ0)
       10
       14
       16
       19
       21
 6 rows selected.
```

In the previous example, the JOB_QUEUE_PROCESSES parameter corresponds to the CJQ0 process entry. Because the job queue process is a user process, it is counted as a SQL session that prevents switchover from taking place. The entries with no process or program information are threads started by the job queue controller.

Verify that the JOB_QUEUE_PROCESSES parameter is set using the following SQL statement:

```
SQL> SHOW PARAMETER JOB_QUEUE_PROCESSES;
NAME                           TYPE      VALUE
------------------------------ -------   --------------------
job_queue_processes            integer   5
```

Then, set the parameter to 0. For example:

```
SQL> ALTER SYSTEM SET JOB_QUEUE_PROCESSES=0;
Statement processed.
```

Because JOB_QUEUE_PROCESSES is a dynamic parameter, you can change the value and have the change take effect immediately without having to restart the instance. You can now retry the switchover procedure.

Do not modify the parameter in your initialization parameter file. After you shut down the instance and restart it after switchover has completed, the parameter will be reset to the original value. This applies to both primary and physical standby databases.

Table A–1 summarizes the common processes that prevent switchover and what corrective action you need to take.

***Table A–1 Common Processes That Prevent Switchover***

| Type of Process | Process Description | Corrective Action |
|---|---|---|
| CJQ0 | The Job Queue Scheduler Process | Change the `JOB_QUEUE_PROCESSES` dynamic parameter to the value 0. The change will take effect immediately without having to restart the instance. |
| QMN0 | The Advanced Queue Time Manager | Change the `AQ_TM_PROCESSES` dynamic parameter to the value 0. The change will take effect immediately without having to restart the instance. |
| DBSNMP | The Oracle Enterprise Manager Intelligent Agent | Issue the `agentctl stop` command from the operating system prompt. |

## A.5 What to Do If SQL Apply Operations to a Logical Standby Database Stop

Log apply services cannot apply unsupported DML statements, DDL statements, and Oracle supplied packages to a logical standby database in SQL apply mode.

When an unsupported statement or package is encountered, SQL apply operations stop. You can take the actions described in Table A–2 to correct the situation and start applying SQL statements to the logical standby database again.

***Table A–2 Fixing Typical SQL Apply Operations Errors***

| If... | Then... |
|---|---|
| You suspect an unsupported statement or Oracle supplied package was encountered | Find the last statement in the DBA_LOGSTDBY_EVENTS view. This will indicate the statement and error that caused SQL apply operations to fail. If an incorrect SQL statement caused SQL apply operations to fail, transaction information, as well as the statement and error information, can be viewed. The transaction information can be used with other Oracle9*i* 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 operations 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 command | Enter the correct SQL statement and use the DBMS_LOGSTDBY.SKIP_TRANSACTION procedure to ensure that the incorrect statement is ignored the next time SQL apply operations are run. Then restart SQL apply operations 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 a DBMS_LOGSTDBY.SKIP('TABLE','schema_name','table_name',null) call, then restart SQL apply operations. |

**See Also:** Chapter 14 for information about querying the DBA_LOGSTDBY_EVENTS view to determine the cause of failures

## A.6 Network Tuning for Redo Log Transmission

The process of archiving redo logs involves reading a buffer from the redo log and writing it to the archive log location. When the destination is remote, the buffer is written to the archive log location over the network using Oracle Net services.

The default archive log buffer size is 1 megabyte. The default transfer buffer size for Oracle Net is 2 kilobytes. Therefore, the archive log buffer is divided into units of approximately 2 kilobytes for transmission. These units could get further divided depending on the maximum transmission unit (MTU) of the underlying network interface.

The Oracle Net parameter that controls the transport size is **session data unit (SDU).** This parameter can be adjusted to reduce the number of network packets that are transmitted. This parameter allows a range of 512 bytes to 32 kilobytes.

For optimal performance, set the Oracle Net SDU parameter to 32 kilobytes for the associated SERVICE destination parameter.

The following example shows a database initialization parameter file segment that defines a remote destination `netserv`:

```
LOG_ARCHIVE_DEST_3='SERVICE=netserv'
SERVICE_NAMES=srvc
```

The following example shows the definition of that service name in the `tnsnames.ora` file:

```
netserv=(DESCRIPTION=(SDU=32768)(ADDRESS=(PROTOCOL=tcp)(HOST=host) (PORT=1521))
(CONNECT_DATA=(SERVICE_NAME=srvc)(ORACLE_HOME=/oracle)))
```

The following example shows the definition in the `listener.ora` file:

```
LISTENER=(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=tcp)
(HOST=host)(PORT=1521))))

SID_LIST_LISTENER=(SID_LIST=(SID_DESC=(SDU=32768)(SID_NAME=sid)
(GLOBALDBNAME=srvc)(ORACLE_HOME=/oracle)))
```

If you archive to a remote site using high-latency / high-bandwidth connections, you can improve performance by increasing the TCP send and receive window sizes. Use caution, however, because this might adversely affect networked applications that do not exhibit the same characteristics as archiving. This method consumes a large amount of system resources.

**See Also:** *Oracle9i Net Services Administrator's Guide*

You can also use cascade standby databases to off-load network processing from the primary database to a standby database. See Appendix D for more information.

## A.7 Managing Data Guard Network Timeout

For any given Oracle Data Guard network connection, there are two processes communicating with each other. When the network connection is unexpectedly broken, how these processes react differs greatly. This is a discussion of what actually occurs when a network connection is broken, and how it affects the Data Guard environment and configuration. This discussion applies to both physical and logical standby databases.

Data Guard uses a peer-to-peer connection protocol, whereby a primary database process, if it is the Log Writer (LGWR) or the Archiver (ARCH), establishes a network connection to the standby database. As a result of the network connection request, the listener on the standby site creates a separate process on the standby database - called the Remote File Server (RFS) process. This RFS process uses network messages from the primary database; it reads from the network and sends an acknowledgement message back to the primary when it is done processing the request.

During normal Data Guard operations, when redo data is transmitted from the primary to the standby, network messages are initiated from the primary database (the network 'client'), and always acknowledged by the standby database (the network 'server'). In this case, the LGWR and ARCH processes are the network clients, and the RFS process is the network server.

Consider the simple scenario where the network between the primary and standby systems is disconnected. This results in what is known as a dead connection. A dead connection indicates that there is no physical connection, but the connection appears to still be there to the processes on each system.

When the LGWR process attempts to send a new message to the RFS process over the dead connection, the LGWR process receives an error from Oracle Net, after a TCP timeout, indicating that the connection has been broken. In this way, the LGWR is able to establish that network connectivity has been lost, and take corrective action. The Data Guard attributes `[NO]MAX_FAILURE`, `[NO]REOPEN` and `[NO]NET_TIMEOUT`, which are options for the `LOG_ARCHIVE_DEST_`*n* parameter, provide LGWR with the desired flexibility to control the timeout intervals and number of retries associated with a network connection that is not responding.

In contrast to the LGWR process, the RFS process on the standby database is always synchronously waiting for a new message to arrive from the primary database. The RFS process that is doing the network read operation is blocked until some data arrives to its reading buffer, or until the underlying network software determines the dead connection is no longer valid.

Oracle Net periodically sends a network probe to verify that a client/server connection is still active. This ensures that connections are not left open indefinitely due to an abnormal client termination. If the probe finds a dead connection or a connection that is no longer in use, it returns an error that causes the RFS process to exit.

You can use the Oracle Net parameter `SQLNET.EXPIRE_TIME` to specify the time interval, expressed in seconds, when to send a probe to verify that the network session is active. Setting this parameter to a small value allows for more timely

detections of dead connections. Connections that do not respond to this probe signal are disconnected. This parameter should be set up for the standby database, as well as the primary, to prepare it for future switchover scenarios.

Limitations on using the dead connection detection feature are:

- Though very small, a probe packet generates additional traffic. However, compared to the network traffic generated by Data Guard that is based on the primary database workload, this additional packet traffic is insignificant.
- Depending on which operating system is in use, the server might need to perform additional processing to distinguish the connection-probing event from other events that occur. This can affect network performance.

Once the RFS process receives notification of the dead network connection, it will terminate itself. However, until such time as the RFS process terminates itself, it will retain lock information on the archivelog on the standby site, or the standby redo log, whose redo information was being received from the primary database. During this interval, no new RFS processes can receive redo information from the primary database for the same archived redo log (or the standby redo log).

The dead network connection detection timer expiration value can also be controlled using the TCP/IP keepalive parameter that specifies the number of seconds to wait before verifying the network connection is valid. Note that the value of the TCP/IP keepalive parameter defaults on most system to two hours, which means that in the default case the RFS process will wait for 2 hours before timing out on a dead network connection.

Therefore, Oracle Corporation recommends setting the Oracle Net `SQLNET.EXPIRE_TIME` parameter and the TCP/IP keepalive parameter to 60 seconds. This is a reasonable value for most systems, and setting the parameter to a small value does not significantly impact production systems.

Once the network problem is resolved, and the primary database processes are again able to establish network connections to the standby database, a new RFS process will automatically be spawned on the standby database for each new network connection. These new RFS processes will resume the reception of redo data from the primary database.

# B

# Manual Recovery

Although Oracle Corporation recommends that you use the managed recovery mode for your physical standby databases, you can also use **manual recovery mode.** You might choose manual recovery mode for any of the following reasons:

- Manual recovery mode allows you to have more than 10 standby databases, which is the limit with managed recovery operations.
- If you do not want to connect to the primary and standby databases over the Oracle Net network, then open the database in manual recovery mode.
- If you are performing managed recovery operations and, for some reason, managed recovery operations no longer work, you can change to manual recovery mode.

This appendix explains how to work in manual recovery mode. It includes the following topics:



## B.1 Preparing a Standby Database for Manual Recovery: Basic Tasks

Table B–1 summarizes the basic tasks for setting up a standby database in preparation for manual recovery. This procedure assumes that you plan to connect to the standby database through Oracle Net. If you do not want to use Oracle Net to connect to the standby database, skip steps 4 and 5.

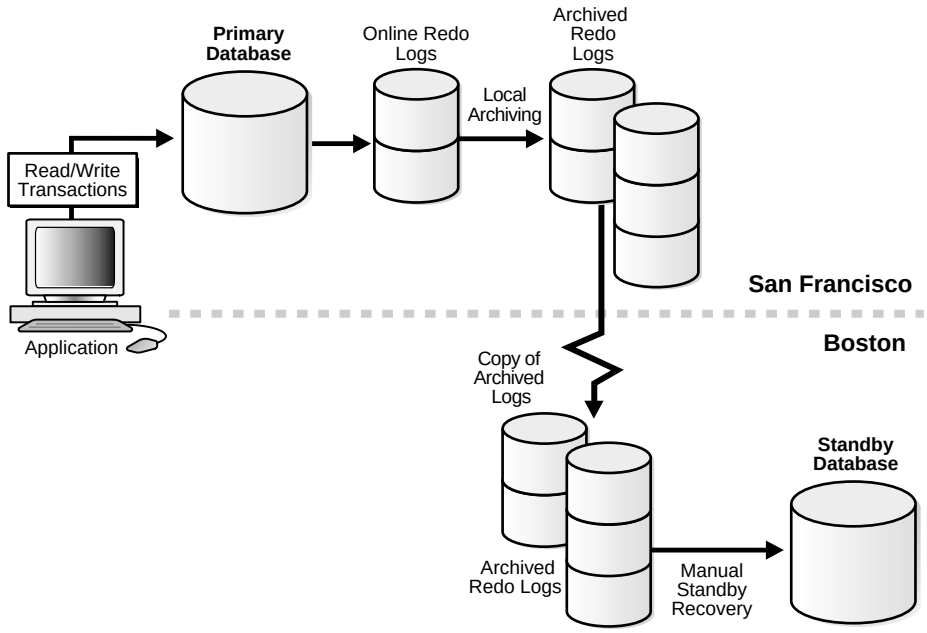
*Table B–1 Task List: Preparing for Manual Recovery*

| Step | Task | Procedure |
|---|---|---|
| 1 | Either make a new backup of the primary database datafiles or access an old backup. | Section 3.2.2 |
| 2 | Connect to the primary database and create the standby control file. | Section 3.2.3 |
| 3 | Prepare and copy the backup datafiles and standby control file from the primary site to the standby site. | Section 3.2.4 and Section 3.2.5 |
| 4 | If you want to create an Oracle Net connection to the standby database, create a service name. | Section 6.4 and LOCATION and SERVICE in Chapter 12 |
| 5 | If you want to create an Oracle Net connection to the standby database, configure the listener on the standby site so that it can receive requests for connections to the standby instance. | Section 6.4 |
| 6 | Create the standby initialization parameter file on the standby site and set the initialization parameters for the standby database. Optionally, set `DB_FILE_NAME_CONVERT` and `LOG_FILE_NAME_CONVERT` to automatically rename primary files in the standby control file. | Section 5.8 |
| 7 | Start the standby instance and mount the standby database. | Section 6.2.1 |
| 8 | While connected to the standby database, manually change the names of the primary datafiles and redo logs in the standby control file for all files *not* automatically renamed using `DB_FILE_NAME_CONVERT` and `LOG_FILE_NAME_CONVERT` in step 6. If step 6 renamed all files, skip this step. | Section B.4 |

## B.2 Placing the Standby Database in Manual Recovery Mode

After you have started and mounted the standby database, you can place it in manual recovery mode. To keep the standby database current, you must manually apply archived redo logs from the primary database to the standby database. Figure B–1 shows a database in manual recovery mode.

***Figure B–1 Standby Database in Manual Recovery Mode***




This section contains the following topics:

- Initiating Manual Recovery Mode
- When Is Manual Recovery Required?

## B.2.1 Initiating Manual Recovery Mode

Archived redo logs arrive at the standby site in one of the following ways:

- The primary database automatically archives the logs (only if you implement a Data Guard environment).
- You manually transfer logs using an operating system utility or some other means.

The standby database assumes that the archived log group is in the location specified by either of the following parameters in the standby initialization parameter file:

- First valid disk location specified by LOG_ARCHIVE_DEST_*n* (where *n* is an integer from 1 to 10)
- Location specified by LOG_ARCHIVE_DEST_*n*

If the archived logs are not in the location specified in the initialization parameter file, you can specify an alternative location using the FROM option of the RECOVER statement.

### To place the standby database in manual recovery mode

1. Use SQL*Plus to connect to the standby instance and then start the Oracle instance at the standby database. For example, enter:

   ```
   STARTUP NOMOUNT pfile=initSTANDBY.ora
   ```

2. Mount the standby database:

   ```
   ALTER DATABASE MOUNT STANDBY DATABASE;
   ```

3. If log transport services are not archiving logs automatically to the standby site, then manually copy the logs to the desired location on the standby site using an appropriate operating system utility for transferring binary data. For example, enter:

   ```
   % cp /oracle/arc_dest/*.arc /standby/arc_dest
   ```

4. Issue a RECOVER statement to place the standby database in manual recovery mode.

   **Note:** Specify the FROM 'location' option only if the archived log group is *not* in the location specified by the LOG_ARCHIVE_DEST_*n* parameter (where *n* is an integer from 1 to 10) or the LOG_ARCHIVE_DEST_*n* parameter in the standby initialization parameter file.

   For example, execute one of the following statements:

   ```
   RECOVER STANDBY DATABASE # uses location for logs specified in
                     # initialization parameter file
   RECOVER FROM '/logs' STANDBY DATABASE # specifies nondefault location
   ```

As the Oracle database server generates archived redo logs, you must continually copy and apply them to the standby database to keep it current.

### B.2.2 When Is Manual Recovery Required?

Manual recovery mode is required in a non-Data Guard environment. A non-Data Guard environment is one in which you manually:

- Transfer the archived redo logs from the primary site to the standby site
- Apply the archived redo logs to the standby database

Even if you implement a Data Guard environment, you might occasionally choose to perform manual recovery on the standby database. For example, you might choose to manually resolve an existing archive gap by using manual recovery mode.

## B.3 Resolving Archive Gaps Manually

An **archive gap** is a range of archived redo logs created whenever you are unable to apply the next archived redo log generated by the primary database to the standby database. This section contains the following topics:

- What Causes Archive Gaps?
- Determining If an Archive Gap Exists
- Manually Transmitting the Logs in the Archive Gap to the Standby Site
- Manually Applying the Logs in the Archive Gap to the Standby Database

> **Note:** Typically, archive gaps are resolved automatically without the need for manual intervention. See Section 6.4 for more information about how log apply services automatically recover from gaps in the redo logs.

### B.3.1 What Causes Archive Gaps?

An archive gap can occur whenever the primary database archives a log, but the log is not archived to the standby site. Because the standby database requires the sequential application of redo logs, media recovery stops at the first missing log encountered.

Archive gaps can occur in the following situations:

- Creation of the Standby Database
- Shutdown of the Standby Database When the Primary Database Is Open

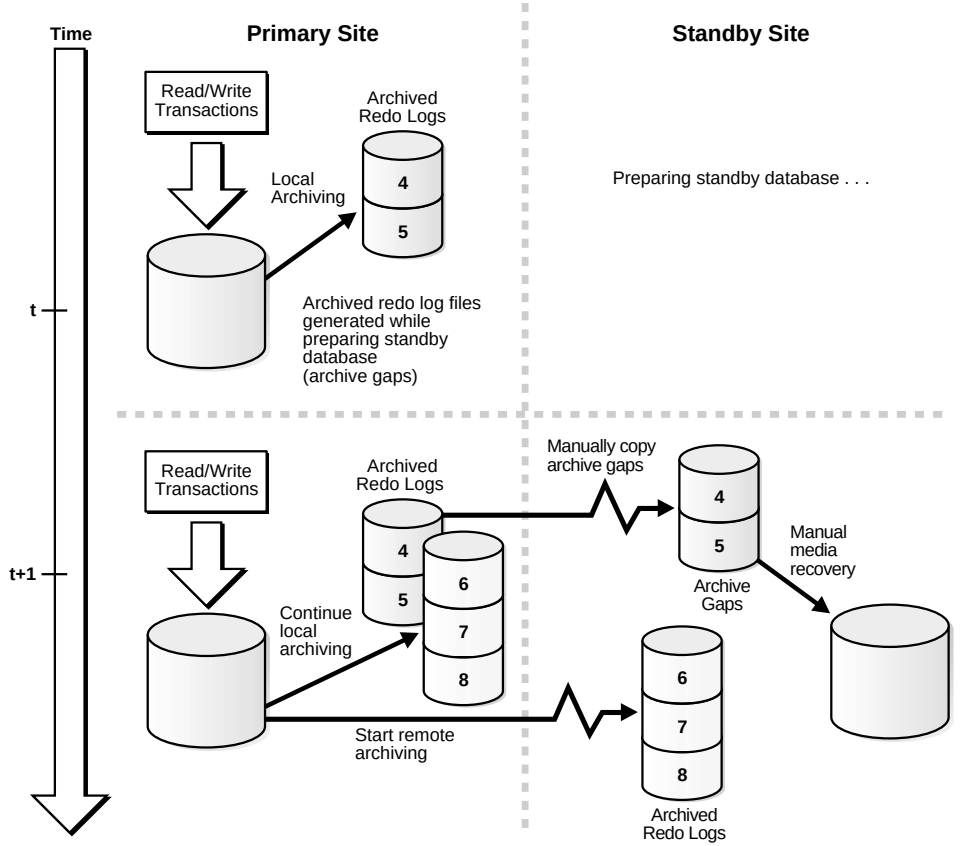
- Network Failure Preventing the Archiving of Logs to the Standby Site

### B.3.1.1 Creation of the Standby Database

One example of an archive gap occurs when you create the standby database from an old backup. For example, if the standby database is made from a backup that contains changes through log 100, and the primary database currently contains changes through log 150, then the standby database requires that you apply logs 101 to 150. Another typical example of an archive gap occurs when you generate the standby database from a hot backup of an open database.

For example, assume the scenario illustrated in Figure B–2.

***Figure B–2 Manual Recovery of Archived Logs in an Archive Gap***



The following steps occur:

1. You take a hot backup of database `primary`.
2. At time *t*, while you are busy configuring the network files, `primary` archives log sequences 4 and 5.
3. At time *t + 1*, you start the standby instance.
4. `primary` archives log sequences 6, 7, and 8 to both the primary site and the standby site.

Archived log sequences 4 and 5 are now part of an archive gap, and these logs must be applied to the standby database.

### B.3.1.2 Shutdown of the Standby Database When the Primary Database Is Open

You might be required to shut down the standby database to resolve maintenance issues. For example, you must shut down the standby database when you change a control file parameter, such as `MAXDATAFILE`, in the primary database.

To avoid creating archive gaps, follow these rules:

- Start the standby databases and listeners *before* starting the primary database.
- Shut down the primary database *before* shutting down the standby database.

If you violate either of these two rules, then the standby database is down while the primary database is open and archiving. Consequently, the Oracle database server can create an archive gap.

---

**Note:** If the standby site is specified as `MANDATORY` in one of the `LOG_ARCHIVE_DEST_`*n* parameters of the primary initialization parameter file, dynamically change it to `OPTIONAL` before shutting down the standby database. Otherwise, the primary database eventually stalls because it cannot archive its online redo logs.

---

### B.3.1.3 Network Failure Preventing the Archiving of Logs to the Standby Site

If you maintain a Data Guard environment, and the network goes down, the primary database might continue to archive to disk but be unable to archive to the standby site. In this situation, archived logs accumulate as usual on the primary site, but the standby instance is unaware of them.

To prevent this problem, you can specify that the standby destination have mandatory status. If the archiving destination is mandatory, then the primary

database will not archive any logs until it is able to archive to the standby site. For example, you can set the following in the primary initialization parameter file to make `standby1` a mandatory archiving destination:

```
LOG_ARCHIVE_DEST_2 = 'SERVICE=standby1 MANDATORY'
```

One consequence of this configuration is that unless the network problem is fixed, the primary database eventually stalls because it cannot switch into an unarchived online redo log. This problem is exacerbated if you maintain only two online redo logs in your primary database.

**See Also:**

- Section 5.4.3 for a detailed account of the significance of the `OPTIONAL` and `MANDATORY` attributes for standby archival
- Section 10.4 for a related scenario

## B.3.2 Determining If an Archive Gap Exists

To determine if there is an archive gap, query the `V$ARCHIVED_LOG` and `V$LOG` views. If an archive gap exists, the output of the query specifies the thread number and log sequence number of all logs in the archive gap. If there is *no* archive gap for a given thread, the query returns no rows.

### To identify the logs in the archive gap

Query the `V$ARCHIVED_LOG` and `V$LOG` views on the standby database. For example, the following query shows that there is a difference in the `RECD` and `SENT` sequence numbers for the destination specified by `DEST_ID=2`, indicating that there is a gap:

```
SQL> SELECT MAX(R.SEQUENCE#) LAST_SEQ_RECD, MAX(L.SEQUENCE#) LAST_SEQ_SENT FROM
  2> V$ARCHIVED_LOG R, V$LOG L WHERE
  3> R.DEST_ID=2 AND L.ARCHIVED='YES';

LAST_SEQ_RECD LAST_SEQ_SENT
------------- -------------
            7            10
```

Use the following query to determine the names of the archived redo logs on the local system that must be copied to the standby system that has the gap:

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

```
NAME
--------------------------------------------------------------------------------

/primary/thread1_dest/arcr_1_7.arc
/primary/thread1_dest/arcr_1_8.arc
/primary/thread1_dest/arcr_1_9.arc
/primary/thread1_dest/arcr_1_10.arc
```

## B.3.3 Manually Transmitting the Logs in the Archive Gap to the Standby Site

After you have obtained the log sequence numbers of the logs in the archive gap, you can obtain their filenames by querying the V$ARCHIVED_LOG view on the primary site. The archived log filenames on the standby site are generated by the STANDBY_ARCHIVE_DEST and LOG_ARCHIVE_FORMAT parameters in the standby initialization parameter file.

If the standby database is on the same site as the primary database, or the standby database is on a remote site with a different directory structure than the primary database, the filenames for the logs on the standby site cannot be the same as the filenames of the logs archived by the primary database. Before transmitting the archived logs to the standby site, determine the correct filenames for the logs at the standby site.

### To copy logs in an archive gap to the standby site

1. Review the list of archive gap logs that you obtained earlier. For example, assume you have the following archive gap:

   ```
   THREAD#    LOW_SEQUENCE#   HIGH_SEQUENCE#
   ---------- -------------   --------------
          1             460              463
          2             202              204
          3             100              100
   ```

   If a thread appears in the view, then it contains an archive gap. You need to copy logs from threads 1, 2, and 3.

2. Determine the filenames of the logs in the archive gap that were archived by the primary database. After connecting to the primary database, issue a SQL query to obtain the name of a log in each thread. For example, use the following SQL statement to obtain filenames of logs for thread 1:

   ```
   SQL> SELECT NAME FROM V$ARCHIVED_LOG WHERE THREAD#=1 AND DEST_ID=1
     2> AND SEQUENCE# > 459 AND SEQUENCE# < 464;
   ```

```
NAME
--------------------------------------------------------------------------
/primary/thread1_dest/arcr_1_460.arc
/primary/thread1_dest/arcr_1_461.arc
/primary/thread1_dest/arcr_1_462.arc
/primary/thread1_dest/arcr_1_463.arc
4 rows selected
```

Perform similar queries for threads 2 and 3.

3. On the standby site, review the settings for STANDBY_ARCHIVE_DEST and LOG_ARCHIVE_FORMAT in the standby initialization parameter file. For example, you discover the following:

```
STANDBY_ARCHIVE_DEST = /standby/arc_dest/
LOG_ARCHIVE_FORMAT = log_%t_%s.arc
```

These parameter settings determine the filenames of the archived redo logs at the standby site.

4. On the primary site, copy the archive gap logs from the primary site to the standby site, renaming them according to values for STANDBY_ARCHIVE_DEST and LOG_ARCHIVE_FORMAT. For example, enter the following copy commands to copy the archive gap logs required by thread 1:

```
% cp /primary/thread1_dest/arcr_1_460.arc /standby/arc_dest/log_1_460.arc
% cp /primary/thread1_dest/arcr_1_461.arc /standby/arc_dest/log_1_461.arc
% cp /primary/thread1_dest/arcr_1_462.arc /standby/arc_dest/log_1_462.arc
% cp /primary/thread1_dest/arcr_1_463.arc /standby/arc_dest/log_1_463.arc
```

Perform similar copy commands to copy archive gap logs for threads 2 and 3.

5. On the standby site, if the LOG_ARCHIVE_DEST and STANDBY_ARCHIVE_DEST parameter values are *not* the same, then copy the archive gap logs from the STANDBY_ARCHIVE_DEST directory to the LOG_ARCHIVE_DEST directory. If these parameter values *are* the same, then you do not need to perform this step.

For example, assume the following standby initialization parameter settings:

```
STANDBY_ARCHIVE_DEST = /standby/arc_dest/
LOG_ARCHIVE_DEST = /log_dest/
```

Because the parameter values are different, copy the archived logs to the LOG_ARCHIVE_DEST location:

```
% cp /standby/arc_dest/* /log_dest/
```

When you initiate manual recovery, the Oracle database server looks at the LOG_ARCHIVE_DEST value to determine the location of the logs.

Now that all required logs are in the STANDBY_ARCHIVE_DEST directory, you can proceed to Section B.3.4 to apply the archive gap logs to the standby database.

**See Also:** Section 6.5.3 and V$ARCHIVED_LOG in Chapter 14

## B.3.4 Manually Applying the Logs in the Archive Gap to the Standby Database

After you have copied the logs in the archive gap to the standby site, you can apply them using the RECOVER AUTOMATIC statement.

### To apply the archived redo logs in the archive gap

1. Start up and mount the standby database (if it is not already mounted). For example, enter:

```
SQL> STARTUP NOMOUNT PFILE=/oracle/admin/pfile/initSTBY.ora
SQL> ALTER DATABASE MOUNT STANDBY DATABASE;
```

2. Recover the database using the AUTOMATIC option:

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

   The AUTOMATIC option automatically generates the name of the next archived redo log needed to continue the recovery operation.

   After recovering the available logs, the Oracle database server prompts for the name of a log that does not exist. The reason is that the recovery process does not know about the logs archived to the standby site by the primary database. For example, you might see:

```
ORA-00308: cannot open archived log '/oracle/standby/standby_logs/arcr_1_
540.arc'
ORA-27037: unable to obtain file status
SVR4 Error: 2: No such file or directory
Additional information: 3
Specify log: {<RET>=suggested | filename | AUTO | CANCEL}
```

3. Cancel recovery after the Oracle database server has applied the available logs, by executing the following statement (or typing CTRL+C):

```
SQL> CANCEL
Media recovery cancelled.
```

The following error messages are acceptable after recovery cancellation and do not indicate a problem:

```
ORA-01547: warning: RECOVER succeeded but OPEN RESETLOGS would get error
below
ORA-01194: file 1 needs more recovery to be consistent
ORA-01110: data file 1: 'some_filename'
ORA-01112: media recovery not started
```

Oracle Corporation recommends automatically applying the logs in the archive gap using the RECOVER MANAGED STANDBY DATABASE clause of the ALTER DATABASE statement.

**See Also:** Section 6.4 for additional information

## B.4 Renaming Standby Database Files Manually

Sometimes all of the primary datafiles and redo logs cannot be renamed in the standby control file by conversion parameters. For example, assume that your database has the following datafiles, which you want to rename as shown in the following table:

| Primary Filename | Standby Filename |
|---|---|
| /oracle/dbs/df1.dbf | /standby/df1.dbf |
| /oracle/dbs/df2.dbf | /standby/df2.dbf |
| /data/df3.dbf | /standby/df3.dbf |

You can set DB_FILE_NAME_CONVERT as follows to convert the filenames for the first two datafiles:

```
DB_FILE_NAME_CONVERT = '/oracle/dbs', '/standby'
```

Nevertheless, this parameter will not capture the renaming of /data/df3.dbf. You must rename this datafile manually in the standby database control file by issuing a SQL statement as follows:

```
SQL> ALTER DATABASE RENAME FILE '/data/df3.dbf' to '/standby/df3.dbf';
```

### To rename a datafile manually

1. Start up and mount the standby database (if it is not already started) and then mount the database:

```
SQL> STARTUP NOMOUNT PFILE=initSTANDBY1.ora;
SQL> ALTER DATABASE MOUNT STANDBY DATABASE;
```

2. Issue an ALTER DATABASE statement for each datafile requiring renaming, where *old_name* is the old name of the datafile as recorded in the control file and *new_name* is the new name of the datafile that will be recorded in the standby control file:

```
SQL> ALTER DATABASE RENAME FILE 'old_name' TO 'new_name';
```

When you manually rename all of the datafiles that are not captured by the DB_FILE_NAME_CONVERT parameter, the standby database control file can correctly interpret the log stream during the recovery process.

# C

# Standby Database Real Application Clusters Support

Oracle9*i* provides the ability to perform true database archiving from a primary database to a standby database when either or both databases reside in a Real Application Clusters environment. This chapter summarizes the configuration requirements and considerations that apply when using Oracle Data Guard with Oracle Real Application Clusters databases. It contains the following sections:

- Configuring Standby Databases in a Real Application Clusters Environment
- Configuration Considerations in Real Application Clusters Environments
- Troubleshooting

## C.1 Configuring Standby Databases in a Real Application Clusters Environment

You can configure a standby database to protect a primary database using Real Application Clusters. The following table describes the possible combinations of instances in the primary and standby databases:

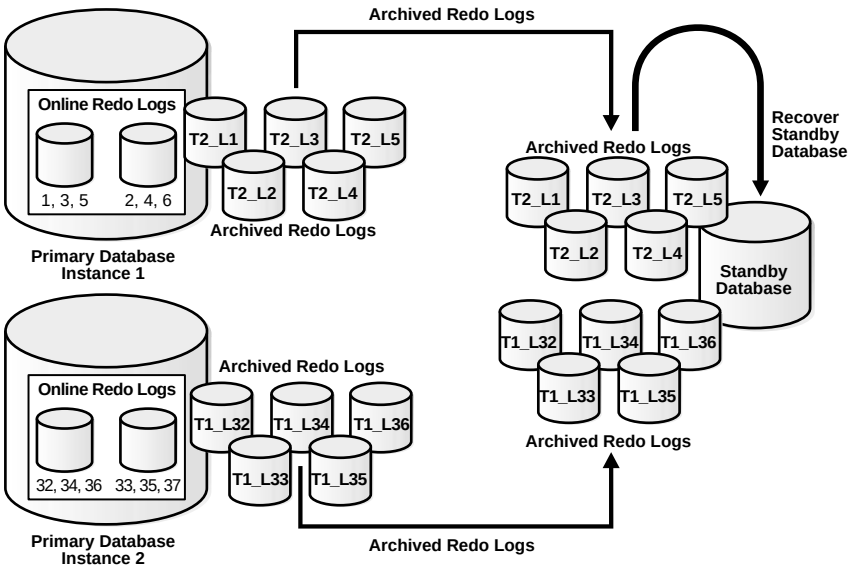
| Instance Combinations | Single-Instance Standby Database | Multi-Instance Standby Database |
|---|---|---|
| **Single-Instance Primary Database** | Yes | Yes (for read-only queries) |
| **Multi-Instance Primary Database** | Yes | Yes |

In each scenario, each instance of the primary database archives its own online redo logs to the standby database.

## C.1.1 Setting Up a Multi-Instance Primary Database with a Single-Instance Standby Database

Figure C–1 illustrates a Real Application Clusters database with two primary database instances (a multi-instance primary database) archiving redo logs to a single-instance standby database.

***Figure C–1 Archiving Redo Logs from a Multi-instance Primary Database***




In this case, Instance 1 of the primary database transmits logs 1, 2, 3, 4, 5 while Instance 2 transmits logs 32, 33, 34, 35, 36. If the standby database is in managed recovery mode, it automatically determines the correct order in which to apply the archived redo logs.

### To set up a primary database in a Real Application Clusters environment

Perform the following steps to set up log transport services on the primary database:

1. On all instances, designate the ARCH or LGWR process to perform the archival operation.
2. Designate the standby database as the receiving node. This is accomplished using the SERVICE attribute of the LOG_ARCHIVE_DEST_*n* initialization parameter.

The standby database also applies the archived redo log it receives through managed recovery to keep itself current with the primary database.

> **See Also:** *Oracle9i Real Application Clusters Setup and Configuration* for information about configuring a database for Real Application Clusters

### To set up a single instance standby database

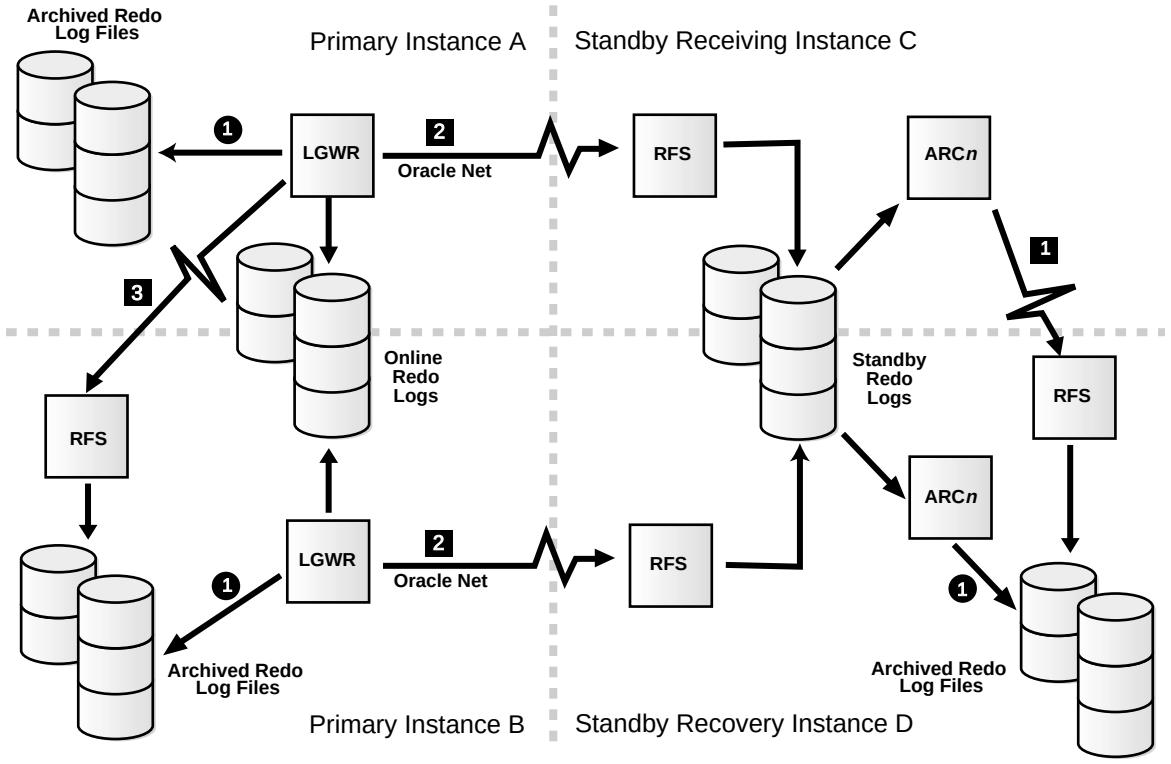
Perform the following steps to set up log transport services on a single instance standby database:

1. Create the standby redo logs if LGWR process is used in log transport services.
2. Define the archived log destination to archive locally if LGWR process is used. This is accomplished using the LOCATION attribute of the LOG_ARCHIVE_DEST_1 initialization parameter. If ARCH process is used in log transport services, define STANDBY_ARCHIVE_DEST and LOG_ARCHIVE_FORMAT to specify the location of archived redo logs.
3. Start the MRP on the standby database.

## C.1.2 Setting Up a Multi-Instance Primary Database with a Multi-Instance Standby Database

This next example shows a configuration where both primary and standby databases are in a Real Application Clusters environment. This allows you to separate the log transport services processing from the log apply services processing on the standby database, thereby improving overall primary and standby database performance. Figure C–2 illustrates a standby database configuration in a Real Application Clusters environment.

***Figure C–2 Standby Database in Real Application Clusters***




In Figure C–2, the numbers within circles indicate local connections, and the numbers within boxes indicate remote connections.

When you use the standby database in a Real Application Clusters environment, any instance can receive archived logs from the primary database; this is the **receiving instance.** However, the archived logs must ultimately reside on disk devices accessible by the node on which the managed recovery operation is performed; this is the **recovery instance.** Transferring the standby database archived logs from the receiving instance to the recovery instance is achieved using the cross-instance archival operation, performed on the standby database.

The standby database cross-instance archival operation requires use of standby redo logs as the temporary repository of primary database archived logs. Using the standby redo logs not only improves standby database performance and reliability, but also allows the cross-instance archival operation to be performed. However,

because standby redo logs are required for the cross-instance archival operation, the primary database must use the log writer process (LGWR) to perform the primary database archival operation.

When both your primary and standby databases are in a Real Application Clusters configuration, and the standby database is in managed recovery mode, then a single instance of the standby database applies all sets of logs transmitted by the primary instances. In this case, the standby instances that are *not* applying redo cannot be in read-only mode while managed recovery is in progress; in most cases, the nonrecoverable instances should be shut down, although they can also be mounted.

### To set up a standby database in a Real Application Clusters environment

Perform the following steps to set up log transport services on the standby database:

1. Create the standby redo logs. In a Real Application Clusters environment, the standby redo logs must reside on disk devices shared by all instances, such as raw devices.
2. On the recovery instance where the managed recovery process (MRP) is to operate, define the archived log destination to archive locally, because cross-instance archiving is not necessary. This is accomplished using the `LOCATION` attribute of the `LOG_ARCHIVE_DEST_1` initialization parameter.
3. On the receiving instance, define the archived log destination to archive to the node where the MRP is to operate. This is accomplished using the `SERVICE` attribute of the `LOG_ARCHIVE_DEST_1` initialization parameter.
4. Start the ARC*n* process on all standby database instances.
5. Start the MRP on the recovery instance.

### To set up a primary database in a Real Application Clusters environment

Perform the following steps to set up log transport services on the primary database:

1. On all instances, designate the LGWR process to perform the archival operation.
2. Designate the standby database as the receiving node. This is accomplished using the `SERVICE` attribute of the `LOG_ARCHIVE_DEST_`*n* initialization parameter.

Ideally, each primary database instance should archive to a corresponding standby database instance. However, this is not required.

### C.1.3 Setting Up a Cross-Instance Archival Database Environment

It is possible to set up a cross-instance archival database environment. Within a Real Application Clusters configuration, each instance directs its archived redo logs to a single instance of the cluster. This instance is called the **recovery instance** and is typically the instance where managed recovery is performed. This instance typically has a tape drive available for RMAN backup and restore support. Example C–1 shows how to set up the LOG_ARCHIVE_DEST_*n* initialization parameter for archiving redo logs across instances. Execute this example on all instances except the recovery instance.

***Example C–1 Setting Destinations for Cross-Instance Archiving***

```
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_1 = 'LOCATION=archivelog MANDATORY REOPEN=120';
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_1 = enable;
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_2 = 'SERVICE=prmy1 MANDATORY REOPEN=300';
SQL> ALTER SYSTEM SET LOG_ARCHIVE_DEST_STATE_2 = enable;
```

Destination 1 is the repository containing the local archived redo logs required for instance recovery. This is a mandatory destination. Because the expected cause of failure is lack of adequate disk space, the retry interval is 2 minutes. This should be adequate to allow the DBA to purge unnecessary archived redo logs. Notification of destination failure is accomplished by manually searching the primary database alert log.

Destination 2 is the recovery instance on the primary database where RMAN is used to back up the archived redo logs from local disk storage to tape. This is a mandatory destination, with a reconnect threshold of 5 minutes. This is the time needed to fix any network-related failures. Notification of destination failure is accomplished by manually searching the primary or standby database alert log.

Cross-instance archiving is available using the ARC*n* process only. Using the LGWR process for cross-instance archiving results in the RFS process failing and the archive log destination being placed in the Error state.

## C.2 Configuration Considerations in Real Application Clusters Environments

This section contains the Data Guard configuration information that is specific to Real Application Clusters environment. It contains the following topics:

- Archived Log File Format
- Archive Destination Quotas

- Data Protection Modes
- Role Transitions

## C.2.1 Archived Log File Format

The format for archive log filenames are usually in the form of log_%parameter where %parameter can be one or more of the following:

| Parameter | Description |
|---|---|
| %T | Thread number, left-zero-padded |
| %t | Thread number, not padded |
| %S | Log sequence number, left-zero-padded |
| %s | Log sequence number, not padded |

For example, `LOG_ARCHIVE_FORMAT = "log_%t_%s.arc"`. The thread parameters%t or %T are mandatory for Real Application Clusters in order to uniquely identify the archived redo logs with the `LOG_ARCHIVE_FORMAT` parameter.

> **See Also:** Section 5.8.4.5 for more information about storage locations for archived redo logs

## C.2.2 Archive Destination Quotas

You can specify the amount of physical storage on a disk device to be available for an archiving destination using the `QUOTA_SIZE` attribute of the `LOG_ARCHIVE_DEST_`*n* initialization parameter. An archive destination can be designated as being able to occupy all or some portion of the physical disk represented by the destination. For example, in a Real Application Clusters environment, a physical archived redo log disk device can be shared by two or more separate nodes (through a clustered file system, such as is available with Sun Clusters). As there is no cross-instance initialization parameter knowledge, none of the Real Application Clusters nodes is aware that the archived redo log physical disk device is shared with other instances. This leads to substantial problems when the destination disk device becomes full; the error is not detected until every instance tries to archive to the already full device. This seriously affects database availability.

## C.2.3 Data Protection Modes

In a Real Application Clusters configuration, any node that loses connectivity with a standby destination will cause all other members of the cluster to stop sending data to that destination (this maintains the data integrity of the data that has been transmitted to that destination and can be recovered).

When the failed standby destination comes back up, Data Guard runs the site in resynchronization mode until the primary and standby databases are identical (no gaps remain). Then, the standby destination can participate in the Data Guard configuration.

The following list describes the effect of the three data protection configurations in Real Application Clusters environment:

- Maximum protection configuration

  If a lost destination is the *last* participating standby site, then the instance on the node that loses connectivity will be shut down. Other nodes in a Real Application Clusters configuration that still have connectivity to the last standby site will recover the lost instance and continue sending to their standby site. Only when every node in a Real Application Clusters configuration loses connectivity to the last standby site will the configuration, including the primary database, be shut down.

  > **Note:** If you are running Real Application Clusters and Data Guard in maximum protection mode and you expect the network to be down for an extended period of time, consider changing the primary database to run in either the maximum availability or the maximum performance mode until network connectivity is restored. See Section C.3.2 for information about changing the primary database temporarily to run in the maximum availability or maximum performance mode.

  When a failover operation occurs to a site that is participating in the maximum protection configuration, all data that was ever committed on the primary database will be recovered on the standby site.

- Maximum availability configuration

  Losing the last standby destination does not cause the primary database instance to shut down.

When a failover operation occurs to a site that is participating in the maximum availability configuration, all data that was ever committed on the primary database and was successfully sent to the standby database will be recovered on the standby site.

- Maximum performance configuration

  Losing the last standby destination does not cause the primary database instance to shut down.

  When a failover operation occurs to any standby site, data that was received from the primary database will be recovered on the standby database up to the last transactionally consistent point in time. In a single-instance configuration, this means all data received will be recovered. In a failover situation, it is possible to lose some transactions from one or more logs that have not yet been transmitted.

## C.2.4 Role Transitions

### C.2.4.1 Switchover Operations

For a Real Application Clusters database, only one primary instance and one standby instance can be active during a switchover operation. Therefore, before a switchover operation, shut down all but one primary instance and one standby instance. After the switchover operation completes, restart the primary and standby instances that were shut down during the switchover operation.

### C.2.4.2 Failover Operations

Before performing a failover to a Real Application Clusters standby database, first shut down all but one standby instance. After the failover operation completes, restart the instances that were shutdown.

If you issue the SQL statement `ALTER DATABASE RECOVER MANAGED STANDBY DATABASE FINISH SKIP STANDBY LOGFILE` to force a standby database into the primary role, log apply services apply archived redo logs until the first unarchived redo log is encountered. All archived redo logs beyond this point are not recovered and all data in them is lost. In a Real Application Clusters environment, use of the `FINISH SKIP STANDBY LOGFILE` clause can result in additional data loss because multiple instances might have dependencies on the redo logs.

# C.3 Troubleshooting

This section provides help troubleshooting problems with Real Application Clusters. It contains the following sections:

- Switchover Fails in a Real Application Clusters Configuration
- Avoiding Downtime in Real Application Clusters During a Network Outage

## C.3.1 Switchover Fails in a Real Application Clusters Configuration

When your database is using Real Application Clusters, active instances prevent a switchover from being performed. When other instances are active, an attempt to switch over fails with the following error message:

```
SQL> ALTER DATABASE COMMIT TO SWITCHOVER TO STANDBY;
ALTER DATABASE COMMIT TO SWITCHOVER TO STANDBY *
ORA-01105: mount is incompatible with mounts by other instances
```

Action: Query the GV$INSTANCE view as follows to determine which instances are causing the problem:

```
SQL> SELECT INSTANCE_NAME, HOST_NAME FROM GV$INSTANCE
  2> WHERE INST_ID <> (SELECT INSTANCE_NUMBER FROM V$INSTANCE);
INSTANCE_NAME HOST_NAME
------------- ---------
INST2         standby2
```

In the previous example, the identified instance must be manually shut down before the switchover can proceed. You can connect to the identified instance from your instance and issue the SHUTDOWN statement remotely, for example:

```
SQL> CONNECT SYS/CHANGE_ON_INSTALL@standby2 AS SYSDBA
SQL> SHUTDOWN;
SQL> EXIT
```

## C.3.2 Avoiding Downtime in Real Application Clusters During a Network Outage

If you configured Data Guard to support a primary database in a Real Application Clusters environment and the primary database is running in maximum protection mode, a network outage between the primary database and all of its physical standby databases will disable the primary database until the network connection is restored. The maximum protection mode dictates that if the last participating physical standby database becomes unavailable, processing halts on the primary database.

If you expect the network to be down for an extended period of time, consider changing the primary database to run in either the maximum availability or the maximum performance mode until network connectivity is restored. If you change the primary database to maximum availability mode, it is possible for there to be a lag between the primary and standby databases, but you gain the ability to use the primary database until the network problem is resolved.

If you choose to change the primary database to the maximum availability mode, it is important to use the following procedures to prevent damage to your data.

Perform the following steps if the network goes down, and you want to change the protection mode for the Real Application Clusters configuration:

1. Shut down the physical standby database.
2. Follow the instructions in Section 5.7 (or see *Oracle9i Data Guard Broker* if you are using the broker) to change the mode from the maximum protection mode to either maximum availability or maximum performance mode.
3. Open the Real Application Clusters primary database for general access.

Later, when the network comes back up, perform the following steps to revert to the maximum protection mode:

1. Shut down the Real Application Clusters primary database, and then mount it without opening it for general access.
2. Mount the physical standby database.
3. Change mode on the Real Application Clusters primary database from its current (maximum availability or maximum performance) mode to the maximum protection mode.
4. Open the Real Application Clusters primary database.

# D

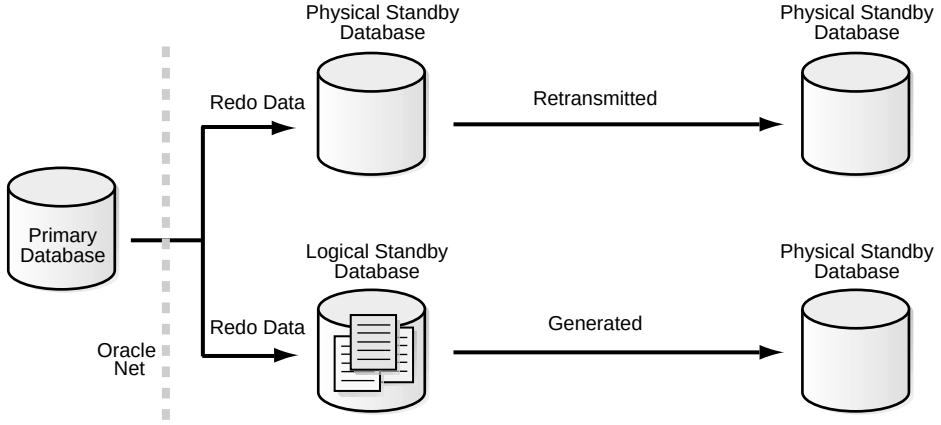
# Cascaded Redo Log Destinations

To reduce the load on your primary system, you can implement **cascaded redo log destinations**, whereby a standby database receives its redo logs from another standby database, instead of directly from the primary database. You can configure:

- A physical standby database to retransmit the incoming redo logs it receives from the primary database to other remote destinations in the same manner as the primary database, with up to one level of redirection.
- A logical standby database (because it is open in read / write mode) to send the redo logs it generates (after filtering and applying the redo data it receives from the primary database) to its own set of standby (physical or logical) databases.

Figure D–1 shows cascaded redo logs to physical and logical standby databases.

***Figure D–1 Cascaded Redo Log Destination Configuration Example***



A standby database can cascade redo data to up to nine destinations. This means that a Data Guard configuration potentially can contain up to 90 standby databases: 1 primary database with 9 standby databases, each of which in turn cascades redo data to 9 additional standby databases. However, from a practical perspective, only standby databases primarily intended to off-load reporting or backup operations typically would be configured to receive cascaded redo data. Standby databases that could potentially be involved in role transition operations typically are configured to receive redo data directly from the primary database and have LOG_ARCHIVE_DEST_*n* and LOG_ARCHIVE_DEST_STATE_*n* parameters defined so that when a switchover or failover operation occurs, logs continue to be received directly from the *new* primary database.

This appendix contains the following sections:



# D.1 Configuring Cascaded Redo Log Destinations

The following sections describe how to set up a Data Guard configuration to use cascaded redo log destinations:



## D.1.1 Configuring Cascaded Redo Log Destinations for Physical Standby Databases

To enable a physical standby database to send the incoming redo logs to another set of destinations, you must define the following items:

- Define the LOG_ARCHIVE_DEST_*n* initialization parameter on the primary database to set up a physical standby database that will be the starting point for a cascade to use the LGWR transport method. Use either SYNC or ASYNC network protocols depending on your requirements.
- On the receiving physical standby database, define sufficient standby redo log files and ensure that archiving is enabled.

At this point, you can begin defining the LOG_ARCHIVE_DEST_*n* initialization parameter on the physical standby database that will define the end points of the cascade. Remember, as part of the original setup of the physical standby database, you should have defined a local archive destination that will be used for local archiving when the physical standby database transitions to the primary role. For

example, you might define the LOG_ARCHIVE_DEST_1 initialization parameter to be the 'LOCATION=/physical1/arch' location. When the physical standby database switches roles, any archived redo logs will be put into that directory with the same format that you defined with the LOG_ARCHIVE_FORMAT initialization parameter. This local archiving destination can be the same as the one defined in the parameter STANDBY_ARCHIVE_DEST, but this is not required.

A side effect of this configuration is that the archive process on the standby database will now try to send the logs not only to the cascading end points but also to the other standby databases and the primary database if they are defined and enabled. This is not a problem, because the receiving database will either reject it if it is the primary database or a standby database that has already received the same log successfully. If the destination is another standby database and it has not received the log successfully, then this acts as an active gap resolution. You can avoid this by setting the state to DEFER for any destinations not involved in the cascade. However, you will have to remember to enable them again if you do a switchover or failover operation.

If you want to have one initialization parameter file to handle both the cascaded redo log destinations and the original primary and standby destinations, define the destinations for the primary database and other standby databases as well as the cascading standby databases. However, the total remote destinations still cannot exceed 10, including the local archiving destination.

Because it is the archiver process and not the log writer process that (during the archiving of the standby online redo logs) is sending the redo information to the cascaded redo log destinations, you are limited to one set of cascaded redo log destinations per standby database that is connected directly to the primary database.

### D.1.2 Configuring Cascaded Redo Log Destinations for Logical Standby Databases

A logical standby database that receives redo data directly from the primary database can be configured to cascade the redo data it generates (after it has filtered and applied the redo data it receives from the primary database) to other standby databases. Because redo data cascaded from a logical standby database is not identical to the redo data originally generated by the primary database, it cannot be applied to any standby database instantiated directly from the primary database. Instead, any standby databases that receives cascaded redo data from a logical standby database must be created from a copy of the logical standby database, and the following will be true:

- Physical standby databases created from a logical standby database will be a block-for-block copy of the logical standby database and a logical copy of the original primary database.
- Logical standby databases created from a logical standby database will be logical copies of the parent logical standby database and might bear only a partial resemblance to the original primary database. This is because the original primary database's data is there but so is anything else stored in the parent logical standby database as well as any other changes such as different indexes or materialized views.

For standby databases that receive cascaded redo data from a logical standby database, you must perform the same setup tasks as for a physical or logical standby database that receives redo data directly from the primary database. You can use any transport mode (LGWR or ARCH) and network protocol (SYNC or ASYNC). If you use the LGWR network protocol, you can optionally use standby online redo logs on your physical standby databases.

## D.2 Examples of Cascaded Redo Log Destinations

The following scenarios demonstrate configuration options and uses for cascaded redo log destinations.

### D.2.1 Scenario 1

You have a primary database in your corporate offices, and you want to create a standby database in another building on your local area network (LAN). In addition, you have a legal insurance requirement to keep the redo information and backup copies off-site at a geographically distant location outside of your LAN but on your wide area network (WAN).

You could define two destinations on your primary database so that redo logs could be transmitted to both of these sites, but this would put an extra workload on your primary database throughput due to the network latency of sending the redo logs over the WAN.

To solve this problem, you could define a tight connection between your primary and physical standby databases in your LAN using the LGWR and SYNC network transports and standby online redo logs. This would protect against losing access to the primary database and provide an alternate site for production when maintenance is required on the primary database. The secondary location on the WAN could be serviced by the physical standby database, ensuring that the redo information is stored off-site. Nightly backup operations on the production

database could then be moved to the WAN remote standby database, which removes the requirement to ship tapes to the off-site storage area.

Finally, in a worst case scenario where you lose access to both the primary database and the physical standby database on the LAN, you could fail over to the remote standby database with minimal data loss. If you can gain access to the online redo log of the last standby database from the original standby database, you could recover it on the remote standby database, incurring no data loss.

The only time you would incur problems by sending the information over the WAN is during a switchover or failover operation, when the physical standby database has transitioned to the primary role. However, this configuration would still meet your insurance requirements.

### D.2.2 Scenario 2

You have a primary database in a remote city, and you would like to have access to its data locally for reporting purposes. The primary database already has a standby database set up for failure protection in an off-site location on the LAN. Putting a destination on the primary database to send the information to your site would adversely affect the performance of the primary database.

Solving this problem is similar to the solution that is described in scenario 1, except that you are sending the redo logs to a logical standby database, because you already have a physical standby database. First, ensure that:

- The physical standby database is receiving its redo logs from the log writer of the primary database.
- Standby redo logs are defined and being used.

  If standby redo logs are not defined, you can define them dynamically on the standby database. The standby database will begin using the standby redo logs after the next log switch on the primary database. If the `LGWR` network transport is not being used, you can dynamically set log transport services on the primary database, and the primary database will start using the log writer at the next log switch.

Next, perform the normal setup tasks for a logical standby database. Any of the steps required to prepare to use a logical standby database must be done at the primary location as usual. After the logical standby database is up and running, define your destination parameters on the physical standby database to send the redo logs over the WAN, where they will be applied to the logical standby database.

### D.2.3 Scenario 3

A primary database located in a manufacturing site already is configured with two physical standby databases. One standby database is located on the LAN in another building, and the second standby database is more remotely located on a WAN in the corporate offices. You cannot use cascading standby databases for the standby database on the WAN, because there is a requirement to have two standby databases in no-data-loss mode. Also, the marketing department has requested access to the manufacturing data for sales predictions. The marketing department needs access to the data on a daily basis, and they want to combine sales data with manufacturing data to better understand sales versus the actual manufacturing times.

One solution would be to allow marketing to access a physical standby database in the corporate offices using read-only mode. However, putting the standby database in read-only mode requires stopping managed recovery operations. This means that the physical standby database can only catch up with the primary database at night, while it is still receiving data from the second and third shifts at the manufacturing plant. In addition, the standby database would always be at least 12 hours behind in applying redo logs. You could add another destination to the primary database to send the redo logs to a new logical standby database in the corporate offices. Because the systems used in the corporate office are different for the physical standby database and the proposed logical standby database, you cannot use the `DEPENDENCY` attribute when defining the standby destinations. Because redo logs need to be transmitted over a WAN, it would degrade performance on the primary database to send the redo data twice, which has been deemed to be unacceptable.

Cascaded redo log destinations can solve this problem. To set this up, you would create a logical standby database following the instructions in Chapter 4, but you would also set up the corporate physical standby database to transmit the redo logs over the corporate LAN to the new logical standby database. In this way, the primary database is only sending the data once over the WAN. The logical standby database could then be modified with new materialized views so that the marketing group can manage the data more efficiently. Because the logical standby database is open for read / write operations, the marketing group can add new schemas and load in sales data without affecting performance on the primary database, or the viability and current state of the physical standby database.

### D.2.4 Scenario 4

You have five Sales offices around the world, each with its own primary database. You would like to implement a failure protection strategy for all of them, as well as a way to get timely access to all data with minimal effect on each primary database.

To solve this problem, you would first implement a no-data-loss environment for each of the five offices by creating a physical standby database (with LGWR and SYNC attributes) local to each office. The physical standby databases could be on a LAN or a WAN. Then, create a logical standby database from each of the five primary databases and locate the logical standby databases in your corporate office. However, instead of having log transport services on each of the five primary databases send the redo logs, you would configure each of the five standby databases to send the redo logs to its logical standby database over the WAN. At one logical standby database (or all of them), you would define database links to each of the other logical standby databases and use them to access all of the sales data. If you decide that you do not need all of the information from each of the five primary databases, but only certain tables, you can use the SKIP routines to stop applying data that you do not need on each of the logical standby databases.

### D.2.5 Scenario 5

You have a primary database that is currently protected only by nightly backup operations. You have been told that you must implement a major failure recovery strategy immediately. You have another system of the same hardware type in-house, but it does not have enough power to serve as a standby database for failover purposes, and it does not have enough disks for the entire database. The only other system available to you that is large enough to hold the entire database is too far away to be put on the LAN, and the WAN that connects to it is extremely slow. The deadline for implementing the strategy is well before any network upgrades can occur. Adding a destination (on the primary database) to send the redo logs to the remote location would severely affect performance.

The interim solution to this problem would be to create a physical standby database on the remote system and create a distribution repository on the local smaller system. A distribution repository is comprised of only the standby control file and the standby database online redo logs, not the data files. You would configure the primary database to send the redo information to the repository locally using the log writer process (LGWR) in synchronous mode (SYNC). Because the connection is over the LAN, the effect on performance would be minimal. The repository would then be configured to send the data onwards over the WAN to the real standby database.

The risk with this configuration is that while the primary database has transmitted all of its data to a standby database, it is possible that the repository has not completed sending the data to the remote standby database at the time of a failure at the primary database. In this environment, as long as both systems do not fail at

the same time, the remote standby database should receive all the data sent up to the last log switch. You would have to send the current online redo log manually.

Once the WAN is upgraded to permit a direct connection to the remote standby database, you can either redirect the destination to the repository to point to the remote standby database directly or create a new destination to the remote standby database and continue transmitting to the repository as an archive log repository.

# E

# Sample Disaster Recovery ReadMe File

In a multiple standby database configuration, you cannot assume that the database administrator (DBA) who set up the multiple standby database configuration is available to decide which standby database to fail over to in the event of a disaster. Therefore, it is imperative to have a disaster recovery plan at each standby site, as well as at the primary site. Each member of the disaster recovery team needs to know about the disaster recovery plan and be aware of the procedures to follow.

Example E–1 shows the kind of information that the person who is making the decision would need when deciding which standby database should be the target of the failover operation.

A ReadMe file is created and maintained by the DBA and should describe how to:

- Log on to the local database server as a DBA
- Log on to each system where the standby databases are located

  There might be firewalls between systems. The ReadMe file should include instructions for going through the firewalls.
- Log on to other database servers as a DBA
- Identify the most up-to-date standby database
- Perform the standby database failover operation
- Configure network settings to ensure that client applications access the new primary database instead of the original primary database

***Example E–1 Sample Disaster Recovery ReadMe File***

```
---------------Standby Database Disaster Recovery ReadMe File---------------

Warning:
```

```
********************************************************************************
Perform the steps in this procedure only if you are responsible for failing over
to a standby database after the primary database fails.

If you perform the steps outlined in this file unnecessarily, you might corrupt
the entire database system.
********************************************************************************

Multiple Standby Database Configuration:

No.     Location       Type     IP Address
--- --------------- --------- --------------
 1   San Francisco   Primary   128.1.124.25
 2   San Francisco   Standby   128.1.124.157
 3   Boston          Standby   136.132.1.55
 4   Los Angeles     Standby   145.23.82.16
 5   San Francisco   Standby   128.1.135.24

You are in system No. 3, which is located in Boston.

Perform the following steps to fail over to the most up-to-date and available
standby database:

1. Log on to the local standby database as a DBA.

    a)  Log on with the following user name and password:

                username: Standby3
                password: zkc722Khn

    b)  Invoke SQL*Plus as follows:

        % sqlplus

    c)  Connect as the DBA as follows:

        CONNECT sys/s23LsdIc AS SYSDBA

2.  Connect to as many remote systems as possible. You can connect to a maximum
    of four systems. System 4 does not have a firewall, so you can connect to it
    directly. Systems 1, 2, and 5 share the same firewall host.  You need to go
    to the firewall host first and then connect to each system.  The IP address
    for the firewall host is 128.1.1.100.  Use the following user name and
    password:
                username: Disaster
```

password: 82lhsIW32

3. Log on to as many remote systems as possible with the following user names and passwords:

   Login information:

| No. | Location | IP Address | username | password |
|---|---|---|---|---|
| 1 | San Francisco | 128.1.124.25 | Oracle9i | sdd290Ec |
| 2 | San Francisco | 128.1.124.157 | Standby2 | ei23nJHb |
| 3 | | (L o c a l) | | |
| 4 | Los Angeles | 145.23.82.16 | Standby4 | 23HHoe2a |
| 5 | San Francisco | 128.1.135.24 | Standby5 | snc#$dnc |

4. Invoke SQL*Plus on each remote system you are able to log on to as follows:

   ```
   % sqlplus
   ```

5. Connect to each remote database as follows:

   ```
   CONNECT sys/password AS SYSDBA
   ```

   The DBA passwords for each location are:

| No. | Location | Password |
|---|---|---|
| 1 | San Francisco | x2dwlsd91 |
| 2 | San Francisco | a239s1DAq |
| 3 | (L o c a l) | |
| 4 | Los Angeles | owKL(@as23 |
| 5 | San Francisco | sad_KS13x |

6. If you are able to log on to System 1, invoke SQL*Plus and execute the following statements:

   ```
   SQL> SHUTDOWN IMMEDIATE;
   SQL> STARTUP PFILE=PRMYinit.ora;
   ```

   Note: If you are able to execute the STARTUP statement successfully, the primary database has not been damaged. Do not continue with this procedure.

7. Execute the following SQL statements on each standby database (including the one on this system) that you were able to connect to:

```
    SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE CANCEL;
    SQL> SELECT THREAD#, MAX(SEQUENCE#) FROM V$LOG_HISTORY GROUP BY THREAD#;

    Compare the query results of each standby database. Fail over to the
    standby database with the largest sequence number.

8.  Fail over to the standby database with the largest sequence number.

    On the standby database with the largest sequence number, invoke SQL*Plus
    and execute the following SQL statements:

    SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE
      2> DISCONNECT FROM SESSION;
    SQL> ALTER DATABASE RECOVER MANAGED STANDBY DATABASE FINISH;
    SQL> ALTER DATABASE COMMIT TO SWITCHOVER TO PRIMARY;
    SQL> SHUTDOWN IMMEDIATE;
    SQL> STARTUP PFILE=Failover.ora;

9.  Update the other standby databases with the new primary database information
and ensure the log transport and apply services are working correctly.

------------End of Standby Database Disaster Recovery ReadMe File-------------
```

# Glossary

### ARCH

*See* archiver process (ARCn)

### archive gap

A range of archived redo logs created whenever you are unable to apply the next archived redo log generated by the primary database to the standby database.

### archived redo log

A copy of one of the filled members of an online redo log group made when the database is in ARCHIVELOG mode. As the LGWR process fills each online redo log with redo records, log transport services copy the log to one or more offline archive log destinations. This copy is the archived redo log, also known as the *offline redo log*.

*See also* ARCHIVELOG mode, online redo log, *and* redo log

### ARCHIVELOG mode

The mode of the database in which log transport services archive filled online redo logs to disk. Specify the mode at database creation or by using the SQL ALTER DATABASE ARCHIVELOG statement. You can enable automatic archiving either dynamically using the SQL ALTER SYSTEM ARCHIVE LOG START statement or by setting the initialization parameter LOG_ARCHIVE_START to TRUE.

Running your database in ARCHIVELOG mode has several advantages over NOARCHIVELOG mode. You can:

- Back up your database while it is open and being accessed by users
- Recover your database to any desired point in time

To protect your database that is in ARCHIVELOG mode in case of failure, back up your archived logs.

*See also* archived redo log, NOARCHIVELOG mode, *and* redo log

### archiver process (ARC*n*)

On the primary database location, the process (or a SQL session performing an archival operation) that creates a copy of the online redo logs, either locally or remotely, for standby databases. On the standby database location, the ARC*n* process archives the standby redo logs to be applied by the managed recovery process (MRP).

### archiving

The operation in which the ARC*n* background process copies filled online redo logs to offline destinations. You must run the primary database in ARCHIVELOG mode to archive redo logs.

### ARC*n*

*See* archiver process (ARCn)

### availability

The measure of the ability of a system or resource to provide the desired service when required. Measured in terms of the percentage of time the device is accessible out of the total time it is needed. Businesses that require uninterrupted computing services have an availability goal of 100%, or 24x365. The sum of availability and downtime equals 100%.

*See also* downtime *and* switchover

### backup control file

A backup of the control file. Make the backup by:

- Using the Recovery Manager utility (RMAN) `backup` or `copy` command. Never create a backup control file by using operating system commands.
- Using the SQL statement `ALTER DATABASE BACKUP CONTROLFILE TO 'filename'`.

Typically, you restore backup control files when all copies of the current control file are damaged; sometimes you restore them before performing certain types of point-in-time recovery.

*See also* control file *and* current control file

### backup piece

A physical file in a format specific to RMAN. The file belongs to only one backup set. A backup set usually contains only one backup piece. The only time RMAN creates more than one backup piece is when you limit the piece size using the `MAXPIECESIZE` option of the RMAN `ALLOCATE` or `CONFIGURE` command.

*See also* backup set *and* Recovery Manager (RMAN)

### backup set

An RMAN-specific logical grouping of one or more physical files called *backup pieces*. The output of the RMAN `BACKUP` command is a backup set. Extract the files in a backup set by using the RMAN `RESTORE` command. You can multiplex files into a backup set, that is, intermingle blocks from input files into a single backup set.

There are two types of backup sets:

- Datafile backup sets, which are backups of any datafiles or a control file. This type of backup set is compressed, which means that it only contains datafile blocks that have been used; unused blocks are omitted.
- Archive log backup sets, which are backups of archived redo logs.

*See also* backup piece *and* Recovery Manager (RMAN)

### broker

A distributed management framework that automates and simplifies most of the operations associated with the creation, control, and monitoring of a Data Guard configuration. The broker includes two user interfaces: Oracle Data Guard Manager (a graphical user interface) and the Data Guard command-line interface.

### cascading standby database

A standby database that receives its redo logs from another standby database, not from the original primary database.

### child destination

A log transport services archiving destination that is configured to receive redo logs from the primary database and depends on the successful completion of archival operations for the parent destination.

*See also* destination dependency, destinations, *and* parent destination

### closed backup

A backup of one or more database files taken while the database is closed. Typically, a closed backup is also a whole database backup (a backup of the control file and all datafiles that belong to a database). If you closed the database cleanly, then all the files in the backup are consistent. If you shut down the database using a `SHUTDOWN ABORT` statement or the instance terminated abnormally, then the backups are inconsistent.

*See also* consistent backup

### cold backup

*See* closed backup

### consistent backup

A whole database backup (a backup of the control file and all datafiles that belong to a database) that you can open with the `RESETLOGS` option without performing media recovery. In other words, you do not need to apply redo logs to datafiles in this backup for it to be consistent. All datafiles in a consistent backup must:

- Have the same checkpoint system change number (SCN) in their headers, unless they are datafiles in tablespaces that are read-only or offline normal (in which case they will have a clean SCN that is earlier than the checkpoint SCN)
- Contain no changes past the checkpoint SCN
- Match the datafile checkpoint information stored in the control file

You can only make consistent backups after you have made a clean shutdown of the database. The database must not be opened until the backup has completed.

*See also* closed backup

### control file

A binary file associated with a database that maintains the physical structure and timestamps of all files in that database. The Oracle database server updates the control file continuously during database use and must have it available for writing whenever the database is mounted or open.

*See also* backup control file *and* current control file

### cross-instance archival environment

The environment in which each instance in a Real Application Clusters configuration directs its archived redo logs to a single instance of the cluster. This single instance is known as the *recovery instance*.

*See also* recovery instance

### current control file

The control file on disk for a primary database; it is the most recently modified control file for the current incarnation of the database. For a control file to be considered current during recovery, it must not have been restored from backup.

*See also* backup control file *and* control file

### current online redo log

The online redo log in which the LGWR background process is currently logging redo records. Those logs to which LGWR is not writing are called inactive.

When LGWR gets to the end of the log, it performs a log switch and begins writing to a new log. If you run the database in `ARCHIVELOG` mode, then the ARC*n* process or processes copy the redo data into an archived redo log.

*See also* online redo log *and* redo log

### Data Guard

The management, monitoring, and automation software that works with a production database and one or more standby databases to protect your data against errors, failures, and corruptions that might otherwise destroy your database.

*See also* primary database *and* standby database

### data loss

Data loss occurs when you fail over to a standby database that has not received all redo data from the primary database.

### datafile

A physical operating system file on disk that was created by the Oracle database server and contains data structures such as tables and indexes. A datafile can only belong to one database.

*See also* tablespace

### destination dependency

Configuring log transport services so that archiving redo logs to a specified destination is dependent upon the success or failure of archiving redo logs to another destination.

*See also* child destination, destinations, *and* parent destination

### destinations

Log transport services allow the primary database to be configured to archive redo logs to up to 10 local and remote locations called destinations.

*See also* child destination, destination dependency, *and* parent destination

### downtime

The measure of the inability of a system or resource to provide the desired service when required. Measured in terms of the percentage or amount of time the device is not accessible out of the total time it is needed. The sum of availability and downtime equals 100%.

A period of time for performing routine maintenance tasks, such as hardware and software upgrades and value-added services is *planned downtime*. The computing system is not available for productive operations during such scheduled maintenance tasks.

*See also* availability *and* switchover

### failover

An irreversible role transition in which a standby database transitions to the primary role, and the old primary database is rendered unable to participate in the configuration. Depending on the protection mode under which the old primary database was operating before the failover, there might be no or some data loss during a failover. A failover is typically used only when a primary database becomes incapacitated (for example, due to a system or software failure), and there is no possibility of performing a switchover or of successfully repairing the primary database within a reasonable amount of time.

*See also* role transition *and* switchover

### FAL client

*See* fetch archive log (FAL) client

### FAL server

*See* fetch archive log (FAL) server

### fetch archive log (FAL)

*See* fetch archive log (FAL) client *and* fetch archive log (FAL) server

### fetch archive log (FAL) client

A background Oracle database server process. The initialization parameter for the FAL client is set on the standby database. The FAL client pulls archived redo logs from the primary location and initiates and requests the transfer of archived redo logs automatically when it detects an archive gap on the standby database.

*See also* fetch archive log (FAL) server

### fetch archive log (FAL) server

A background Oracle database server process that runs on the primary or other standby databases and services the fetch archive log (FAL) requests coming from the FAL client. For example, servicing a FAL request might include queuing requests (to send archived redo logs to one or more standby databases) to an Oracle database server that runs the FAL server. Multiple FAL servers can run on the same primary database at one time. A separate FAL server is created for each incoming FAL request. The initialization parameter for the FAL server is set on the standby database.

*See also* fetch archive log (FAL) client

### full supplemental logging

Ensures supplemental logging is set up correctly by performing log switches and then invoking the `DBMS_LOGMNR_D.BUILD` procedure.

*See also* supplemental logging

### hot backup

*See* open backup

### LGWR

*See* log writer process (LGWR)

### listener

An application that receives requests by clients and redirects them to the appropriate server.

### log apply services

The component of the Data Guard environment that is responsible for applying archived redo logs on the standby database to maintain transactional synchronization with the primary database.

*See also* log transport services, role management services, *and* SQL apply mode

### log switch

The point at which LGWR stops writing to the active redo log and switches to the next available redo log. LGWR switches when either the active log is filled with redo records or you force a switch manually.

If you run your database in ARCHIVELOG mode, log transport services archive the redo data in inactive logs into archived redo logs. When a log switch occurs and LGWR begins overwriting the old redo data, you are protected against data loss because the archived redo log contains the old data. If you run in NOARCHIVELOG mode, log transport services overwrite old redo data at a log switch without archiving it. Hence, you lose all old redo data.

*See also* redo log

### log transport services

The component of the Data Guard environment that is responsible for the automated transfer of primary database online redo data. Log transport services provide for the management of archived redo log permissions, destinations, transmission, reception, and failure resolution. In a Data Guard environment, the log transport services component coordinates its activities with the log apply services component.

*See also* log apply services, no data loss, *and* role management services

### log writer process (LGWR)

The background process that collects transaction redo and updates the online redo logs. The log writer process can also create local archived redo logs and transmit online redo logs to standby databases.

### logging

*See* full supplemental logging *and* supplemental logging

### logical standby database

A standby database that is logically identical to the primary database and can be used to take over processing if the primary database is taken offline. A logical standby database is the logical equivalent of the primary database; they share the same schema definition.

*See also* physical standby database *and* standby database

### logical standby process (LSP)

The LSP applies archived redo log information to the logical standby database.

### managed recovery mode

An environment in which the primary database automatically archives redo logs to the standby location, initiated by entering the following SQL statement:

```
ALTER DATABASE RECOVER MANAGED STANDBY DATABASE;
```

When a standby database runs in managed recovery mode, it automatically applies redo logs received from the primary database.

### managed recovery process (MRP)

The process that applies archived redo log information to the standby database.

### managed standby environment

*See* standby database environment

### manual recovery mode

An environment in which the primary database does not automatically archive redo logs to the standby location. In this environment, you must manually transfer archived logs to the standby location and manually apply them by issuing the following SQL statement:

```
ALTER DATABASE RECOVER STANDBY DATABASE;
```

This mode allows you to recover a standby database manually.

### maximum availability mode

A log transport services data availability mode that can be set to ensure that redo logs are available at a standby database before the current database transaction is committed.

*See also* no data loss, log transport services, maximum performance mode, *and* maximum protection mode

### maximum performance mode

A log transport services data availability mode that offers the lowest level of data protection. In this mode, the archiver process writes redo log data asynchronously to a standby database with as little effect as possible on the performance of the primary database. In a failover operation, it is possible to lose data from one or more logs that have not yet been transmitted. This is the default protection mode.

*See also* no data loss, log transport services, maximum availability mode, *and* maximum protection mode

### maximum protection mode

A log transport services data availability mode that can be set to ensure that redo logs are available at a standby database before primary database processing can continue. This mode is not supported on logical standby databases because standby redo logs are required for this mode.

*See also* no data loss, log transport services, maximum availability mode, *and* maximum performance mode

### MRP

*See* managed recovery process (MRP)

### no data loss

A log transport services option that can be configured to ensure that data modifications made to the primary database are not acknowledged until those data modifications are also available on (but not necessarily applied to) the standby database.

*See also* log transport services, maximum availability mode, *and* maximum protection mode

### NOARCHIVELOG mode

The mode of the database in which log transport services do not require filled online redo logs to be archived to disk. Specify the mode at database creation or change it by using the SQL `ALTER DATABASE` statement. Oracle Corporation does not recommend running in `NOARCHIVELOG` mode because it severely limits the possibilities for recovery of lost data.

*See also* ARCHIVELOG mode

### node

*See* site

### non-managed recovery mode

*See* manual recovery mode

### offline redo log

*See* archived redo log

### online redo log

A set of two or more files that records all changes made to datafiles and control files. Whenever a change is made to the database, the Oracle database server generates a redo record in the redo buffer. The LGWR process flushes the contents of the redo buffer into the online redo log.

Every database must contain at least two online redo logs. If you are multiplexing your online redo log, LGWR concurrently writes the same redo data to multiple files. The individual files are called members of an online redo log group.

*See also* archived redo log, current online redo log, redo log, *and* standby redo log

### open backup

A backup of one or more datafiles taken while a database is open. This is also known as a *hot backup*.

### parent destination

A log transport services archiving destination that has a child destination associated with it.

*See also* child destination, destination dependency, *and* destinations

### physical standby database

A standby database that is physically identical to the primary database because recovery applies changes block-for-block using the physical row ID.

*See also* logical standby database *and* standby database

### planned downtime

*See* availability, downtime, *and* switchover

### primary database

In a Data Guard configuration, a production database is referred to as a primary database. A primary database is used to create a standby database. Every standby database is associated with one and only one primary database. A single primary database can, however, support multiple standby databases.

*See also* standby database

### protected mode

*See* maximum protection mode

### read-only database

A database opened with the SQL statement `ALTER DATABASE OPEN READ ONLY`. As their name suggests, read-only databases are for queries only and cannot be modified. A standby database can be run in read-only mode, which means that it can be queried while still serving as an up-to-date emergency replacement for the primary database.

*See also* read-only mode

### read-only mode

A physical standby database mode initiated by issuing the following SQL statement:

```
ALTER DATABASE OPEN READ ONLY;
```

This mode allows you to query the physical standby database, but not to make changes to it.

*See also* read-only database

### receiving instance

When you use a standby database in a Real Application Clusters configuration, any instance can receive archived logs from the primary database; this is the receiving instance.

*See also* recovery instance

### recovery catalog

A set of tables and views used by Recovery Manager (RMAN) to store information about Oracle databases. RMAN uses this data to manage the backup, restoration, and recovery of Oracle databases. If you choose not to use a recovery catalog, RMAN uses the target database control file. You should not store the recovery catalog in your target database.

*See also* recovery catalog database *and* Recovery Manager (RMAN)

### recovery catalog database

An Oracle database that contains a recovery catalog schema.

*See also* recovery catalog

### recovery instance

The node where managed recovery is performed. Within a Real Application Clusters configuration, each primary instance directs its archived redo logs to this node of the standby cluster.

*See also* cross-instance archival environment *and* receiving instance

### Recovery Manager (RMAN)

A utility that backs up, restores, and recovers Oracle databases. You can use it with or without the central information repository called a *recovery catalog*. If you do not use a recovery catalog, RMAN uses the control file of the database to store information necessary for backup and recovery operations. You can use RMAN in conjunction with a media manager, which allows you to back up files to tertiary storage.

*See also* backup piece, backup set, *and* recovery catalog

### redo log

A file containing redo records. There are three types of redo logs: online redo logs, standby redo logs, and archived redo logs.

The online redo log is a set of two or more files that records all changes made to datafiles and control files. The LGWR process records the redo records in the log. The current online redo log is the one to which LGWR is currently writing.

The standby redo log is an optional location where the standby database can store the redo data received from the primary database. This redo data can be stored on the standby location using either standby redo logs or archived redo logs.

The archived redo log, also known as the *offline redo log*, is a copy of the online redo log that has been copied to an offline destination. If the database is in `ARCHIVELOG` mode, the ARC*n* process or processes copy each online redo log to one or more archive log destinations after it is filled.

*See also* archived redo log, ARCHIVELOG mode, current online redo log, log switch, online redo log, *and* standby redo log

### reliability

The ability of a computing system or software to operate without failing.

### remote file server (RFS)

The remote file server process on the standby location receives archived redo logs from the primary database.

### RMAN

*See* Recovery Manager (RMAN)

### role management services

The component of the Data Guard environment that is responsible for the changing of database roles. Database role transitions include switchover and failover if the primary database is unavailable due to an unplanned shutdown.

*See also* log apply services *and* log transport services

### role transition

A database can be in one of two mutually exclusive roles: primary or standby. You can change these roles dynamically as a planned transition (switchover), or you can change these roles as a result of an unplanned failure (failover).

*See also* failover *and* switchover

### rolling upgrade

A software installation technique that allows a clustered system to continue to provide service while the software is being upgraded to the next release. This process is called a rolling upgrade because each database or system in the cluster is upgraded and rebooted in turn, until all databases or systems have been upgraded.

### site

In a Data Guard configuration, this term is sometimes used to refer to the local or geographically remote location of a primary or standby database.

In a Data Guard broker configuration, a site is a managed unit of failover.

### SQL apply mode

The mode in which log apply services automatically apply archived redo log information to a logical standby database by transforming transaction information into SQL statements and then executing the SQL statement to the logical standby database.

*See also* log apply services

### standby database

An identical copy of a primary database that you can use for disaster protection. You can update your standby database with archived redo logs from the primary database to keep it current. Should a disaster destroy or compromise the primary database, you can fail over to the standby database and make it the new primary

database. A standby database has its own initialization parameter file, control file, and datafiles.

*See also* logical standby database, physical standby database, *and* primary database

### standby database environment

The physical configuration of the primary and standby databases. The environment depends on many factors, including the:

- Number of standby databases associated with a primary database
- Number of host systems used by the databases
- Directory structures of the databases
- Network configuration

A configuration in which a primary database automatically archives redo logs to a standby location is a *managed standby environment*. If the standby database is in managed recovery mode, it automatically applies the logs received from the primary database to the standby database. Note that in a managed standby environment, a primary database continues to transmit archived redo logs even if the standby database is not in managed recovery mode.

### standby redo log

The standby redo log is an optional set of logs where the standby database can store the redo data received from the primary database. (Redo data can also be stored on the standby location using archived redo logs.) Standby redo logs are created using the `ADD STANDBY LOGFILE` clause of the SQL `ALTER DATABASE` statement. Additional log group members can be added later to provide another level of reliability against disk failure on the standby location. Standby redo logs are required if you are using the maximum protection mode. Standby redo logs are not supported for logical standby databases.

*See also* redo log

### supplemental logging

The ability to log additional information in the redo log stream to enable LogMiner to group and merge the redo streams related to a row change during log mining, and also to be able to identify the row using an identification key.

*See also* full supplemental logging

### switchover

A reversible role transition between the primary database and one of its standby databases. The primary database and standby database involved in the switchover operation exchange roles with no loss of application data and no need to restart or re-create any of the other standby databases in the configuration. You cannot use a switchover operation to perform a rolling upgrade of Oracle software. However, it might be possible to use a switchover operation to perform a hardware-based rolling upgrade.

*See also* availability, downtime, failover, and role transition

### system change number (SCN)

A stamp that defines a committed version of a database at a point in time. The Oracle database server assigns every committed transaction a unique SCN.

### tablespace

One or more logical storage units into which a database is divided. Each tablespace has one or more physical datafiles exclusively associated with it.

*See also* datafile

### TAF

*See* transparent application failover (TAF)

### target database

In RMAN, the database that you are backing up or restoring.

### tempfile

A file that belongs to a temporary tablespace, and is created with the `TEMPFILE` option. Temporary tablespaces cannot contain permanent database objects such as tables, and are typically used for sorting. Because tempfiles cannot contain permanent objects, RMAN does not back them up.

*See also* temporary tablespace

### temporary tablespace

Tablespace of temporary tables created during the processing of a SQL statement. This allows you to add tempfile entries in read-only mode for the purpose of making queries. You can then perform on-disk sorting operations in a read-only database without affecting dictionary files or generating redo entries.

*See also* tablespace *and* tempfile

**transparent application failover (TAF)**

The ability of client applications to automatically reconnect to a database and resume work after a failover occurs.

# Index

## A

## B

## C

## D

## E



## F



## G



## H

## I



## J



## K



## L

## M

## N

## O



## P

## Q



## R

## S

## T

## U



## V